Institut für Informatik der Universität Freiburg (Schweiz)

Eine statistische Methode zur Erkennung von Dokumentstrukturen

Inaugural-Dissertation

zur Erlangung der Würde eines Doctor Scientiarum Informaticarum der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Freiburg in der Schweiz.

vorgelegt von

Rolf BRUGGER

aus

Veltheim (AG)

Dissertation Nr. 1251
Hausdruckerei, Universität Freiburg
Mai, 1998

Von der Mathematisch-Naturwissenschaftlichen Fakulät der Universität Freiburg in der Schweiz angenommen, auf Antrag von:

- Rolf Ingold, Universität Freiburg, Schweiz;
- Horst Bunke, Universität Bern, Schweiz;
- Peter King, University of Manitoba, Winnipeg, Canada;
- Béat Hirsbrunner, Universität Freiburg, Präsident der Prüfungskommision.

Freiburg, den 31. Mai, 1999

Der Leiter der Dissertation:	Der Dekan:
Prof. Rolf Ingold	Prof. Béat Hirsbrunner

Dank

Wohl jeder, der zum ersten Mal eine Doktorarbeit in Angriff nimmt, sieht sich zuallererst vor eine Aufgabe monumentalen Ausmasses gestellt. Die jahrelangen Forschungen, begleitet von Publikationen und abgeschlossen von einem detaillierten Buch kann eine Einzelperson alleine unmöglich realisieren. Mir war das unverhoffte Glück beschieden, in vielfacher Hinsicht auf die Unterstützung und Hilfe anteilnehmender Kollegen und Freunde zählen zu dürfen. All diesen Personen sei von ganzem Herzen gedankt:

- Rolf Ingold verdanke ich eine besonders intensive Betreuung vom Anfang bis zum Abschluss der Arbeit. Er hat mich das Forschen und Publizieren gelehrt wobei mir sein kritischer Scharfsinn ein stets willkommener Begleiter war.

- Horst Bunke kann man als eine der grossen Kapazitäten des Forschungsbereichs der Dokumenterkennung betiteln. Ihn als Gutachter im Expertengremium zu wissen war mir immer eine grosse Ehre.

- Peter King, als weiteren Gutachter, danke ich besonders für die befruchtenden Diskussionen gegen Ende meiner Forschungen und die wertvollen Anmerkungen zur Dissertationsschrift.

- Robert Steiner hat mit enormen Einsatz und voller Verve als „stilistischer“ Lektor wesentlich zur sprachlichen Fassung des vorliegenden Textes beigetragen.

- Lukas Theiler bin ich zu Dank verpflichtet für seine kritische Lektüre und Anmerkungen zu frühen Fassungen dieses Textes.

- Besonderen Dank möchte ich auch an alle in unserer Arbeitsgruppe richten: an Antoine Azokly, Tao Hu und Abdelwahab Zramdini, auf deren Arbeiten ich aufbauen konnte, Daniel Bollinger für die Entwicklung des DTD-Inferenzalgorithmus und Folco Banfi, Lyse Robadey, Oliver Hitz und

insbesondere Frédéric Bapst für die hilfsbereite und kollegiale Zusammenarbeit.

- Herzlichen Dank auch an die weiteren Mitglieder unseres Instituts, an die Assistenten für die vielen anregenden Diskussionen und an das technische und administrative Personal, das leider viel zu unbeachtet unserem Institut den Schwung erhält.
- Schliesslich danke ich Béatrice, meiner Familie und meinen Freunden für Liebe, Beistand und Zerstreuung.

Inhaltsverzeichnis

Abstract

This PhD thesis is on the topic of document recognition. It particularly discusses the aspects of learning document models and the recognition of the logical structure of documents. In order to achieve high reliability and user friendliness, we describe an interactive system which can easily be adapted to new document classes.

In an initial learning session the system is able to generate a recognition model based on a small set of completely tagged logical documents. In the successive recognition sessions, the user interactively corrects the recognition errors of the system. In order to prevent it from repeating the same errors again, these corrections are automatically integrated to the model thanks to the system's incremental learning capabilities.

The representation of the document model is based on a novel, statistical formalism. It is based on n-grams, which have been generalized to be able to represent tree structures. The basic principle consists in the representation of local patterns in tree structures using the conditional probabilities of n-grams. Such a statistical model is able to represent one document class at a time. In the discussion of the expressiveness of the statistical model, we introduce the notion of the entropy of a model.

We further introduce a learning algorithm, which estimates the n-gram probabilities of the model based on a set of sample documents. The same algorithm is again used in the incremental learning steps.

The recognition of the physical structure of a document is based on classical methods that have been documented in the literature. However, the logical structure tree is here constructed stepwise on top of the physical structure, using a heuristic bottom-up procedure. The optimal solution is found in an efficient way by a quality measure and a best-first search strategy.

The approach has been empirically validated on three different document classes, the main test series consisting in 25 documents of an article collection with average structural complexity and containing a total of 400 pages. The tests revealed that the recognition rate of the system constantly improves with the number of recognized documents. When the end of this training and recognition phase has been reached, about one correction is necessary every four pages.

Finally, possibilities of integrating the statistical n-gram model with existing standards like SGML/DSSSL are discussed. To this purpose, a method which translates a statistical model into the corresponding DTD is described.

Keywords: document recognition, document structure, statistical model, n-gram, heuristics, DTD inference.

Zusammenfassung

Die vorliegende Dissertation behandelt die Erkennung von Dokumenten. Es werden schwerpunktmässig die Aspekte des Lernens von Dokumentmodellen und der Erkennung der logischen Struktur von Dokumenten betrachtet. Um sowohl eine hohe Zuverlässigkeit als auch Bedienungsfreundlichkeit zu erreichen, wird ein interaktives System beschrieben, das sich leicht an neue Dokumentklassen anpassen lässt.

Das System benötigt eine initiale Lernfähigkeit, indem es aus vollständigen, logischen Dokumenten ein vorläufiges Erkennungsmodell generieren kann. In darauf folgenden Erkennungsvorgängen werden allfällige Fehler interaktiv vom Benutzer korrigiert. Durch die inkrementelle Lernfähigkeit des Systems werden die Korrekturen in das Modell integriert, und so die Wiederholung desselben Fehlers verhindert.

Für die Darstellung des Dokumentmodells wird ein neuartiger, statistischer Formalismus verwendet. Er basiert auf n-Grammen, die in einer Weise erweitert wurden, dass sie auch Baumstrukturen repräsentieren können. Das Grundprinzip basiert auf der Darstellung lokaler Muster in Baumstrukturen durch die bedingten Wahrscheinlichkeiten von n-Grammen. Ein derartiges statistisches Modell vermag jeweils eine Dokumentklasse vollständig zu beschreiben. In der Diskussion um die Repräsentationsfähigkeit des statistischen Modells wird der Begriff der Entropie eingeführt.

Es wird ein Lernalgorithmus vorgestellt, der die n-Gramm-Wahrscheinlichkeiten aus vorgelegten Beispieldokumenten schätzt. Derselbe Algorithmus gelangt auch in inkrementellen Lernphasen zur Anwendung.

Die Erkennung der physischen Struktur eines Dokuments erfolgt mit klassischen Methoden aus der einschlägigen Literatur. Auf der physischen Struktur eines zu erkennenden Dokuments wird mit einem bottom-up Verfahren der logische Strukturbaum konstruiert. Die Heuristik wählt unter Verwendung einer Bewertungsfunktion und einer best-first Suchstrategie effizient eine optimale Lösung aus.

Der Ansatz wird an Dokumenten aus drei verschiedenen Klassen validiert. Die Haupttestserie besteht aus 25 Dokumenten mit insgesamt 400 Seiten einer Serie von Artikeln mittlerer Komplexität. Die Tests belegen, dass die Erkennungsleistung des Systems mit der Anzahl erkannter Dokumente zunimmt, so dass schliesslich etwa eine Korrektur pro vier Seiten nötig ist.

Schliesslich werden Integrationsmöglichkeiten des statistischen n-Gramm-Modells mit bestehenden Standards wie zum Beispiel SGML/DSSSL erforscht. Es wird dazu eine Methode vorgestellt, die ein statistisches Modell in eine entsprechende DTD übersetzt.

Schlüsselwörter: Dokumenterkennung, Dokumentstrukturen, statistisches Modell, n-Gramme, Heuristik, Inferenz von DTDs.

Kapitel 1

Einleitung

Für einen Menschen ist es eine leichte Aufgabe, sich in einem Buch zurechtzufinden. Mit Selbstverständlichkeit liest er das Vorwort und die darauf folgenden Kapitel. Er kennt den Titel und den Autor des Buches, oder er weiss zumindest, wo er es nachsehen kann. Ohne Anstrengung liest und interpretiert er das Inhaltsverzeichnis, er kann Begriffe im Index nachschlagen und die entsprechende Textstelle finden. All dies tut er mit Leichtigkeit und meist unbewusst mit den verschiedensten Typen von Dokumenten, sei es ein Kochrezept, eine Bedienungsanleitung oder ein wissenschaftlicher Artikel.

Ist man bestrebt, ähnliche Fähigkeiten zu automatisieren und auf Rechnern zu implementieren, sieht man sich jedoch vor erhebliche Probleme gestellt. Die schiere Vielfalt von Dokumenttypen und Schriftarten, Alphabeten und Darstellungsarten überfordert jedes bekannte System. Es gibt allerdings eine ganze Reihe interessanter Anwendungen für die Dokumenterkennung, und in automatisierten Dokument-Management-Systemen stellt sie gar eine Schlüsseltechnologie dar.

Die hier vorgestellte Arbeit unternimmt einen weiteren Schritt in den Forschungen, die ein intelligentes System für die Erkennung von Dokumenten zum Ziel haben und die in den vergangenen Jahren erhebliche Fortschritte verbuchen konnten.

Was aber heisst *intelligente Dokumenterkennung*, wozu ist sie notwendig und welches sind die Schwierigkeiten, mit denen man dabei konfrontiert wird? Diese Fragen, und worin der Inhalt dieser Arbeit besteht, werden in den folgenden Abschnitten behandelt.

1.1 Allgemeine Problemstellung

1.1.1 Dokumenterkennung

Dokumente dienen als Träger von Information. Mit ihnen lässt sich nahezu beliebig komplexe Information gliedern und darstellen. Sie können vervielfältigt, verteilt und gelagert werden. Ein Dokument ist somit eine Einheit für die Darstellung, Organisation, Verteilung und Archivierung von Information, sei es in textueller oder graphischer Form. Dokumente können auf einer Vielzahl von Medien dargestellt werden. Das Papier ist das klassische und nach wie vor am weitesten verbreitete Medium für Dokumente.

Mit der der Entwicklung der elektronischen Datenverarbeitung aber war es naheliegend, das Konzept des Dokuments auf den Computer zu übertragen. Die Verarbeitung solcher *elektronischer Dokumente* beschränkte sich zunächst auf den Produktionsprozess, der bis heute noch eine der häufigsten Anwendungen des Computers ist.

Der obere Teil der Abbildung 1.1 illustriert den Ablauf der Dokumentproduktion. Mit Hilfe eines Editorprogramms kann ein Autor ein Dokument erstellen und verändern. Er manipuliert in dieser Phase der Produktion die abstrakteste Repräsentation: das *logische* Dokument. Nur auf dieser Abstraktionsebene lässt sich der Inhalt eines Dokuments bequem ändern.

Durch den Prozess der *Formatierung* gelangt man zum *physischen* Dokument. Hierbei erhält das Dokument ein Layout, indem ein Satzprogramm den Wörtern und Zeichen Schriftarten zuweist und die Spaltenaufteilung, Zeilen- und Seitenumbrüche unter Berücksichtigung typographischer Regeln berechnet. Die Formatierung in ein physisches Dokument ist kein eindeutiger Vorgang. Ein und dasselbe logische Dokument lässt sich in unzählige physische Dokumente übersetzen, abhängig vom gewählten Satzspiegel und Typographieregeln. Sowohl der Inhalt als auch das Layout eines Dokuments sind auf der physischen Ebene grundsätzlich nicht mehr veränderbar. Die wesentliche Eigenschaft des physischen Dokuments ist seine Unabhängigkeit von Ausgabemedien und der Plattform bei der Ausgabe auf ein physisches Medium.

Die tiefste Abstraktionsstufe stellt das *Dokumentbild* dar, das durch Rastern oder *Rendering* aus dem physischen Dokument berechnet wird; üblicherweise wird die Bildauflösung dem Ausgabemedium entsprechend angepasst.

Eine detaillierte Diskussion der Abstraktionsstufen von Dokumenten erfolgt im Kapitel 2 über Dokumentstrukturen.

Die Dokumenterkennung begeht den umgekehrten Weg der Produktion (siehe unterer Teil der Abbildung 1.1). Das Ziel ist die Transformation der meistens auf Papier vorhandenen und für den Menschen verständliche Information in eine durch den Computer verarbeitbare Form. Durch *Scannen* wird ein Dokument

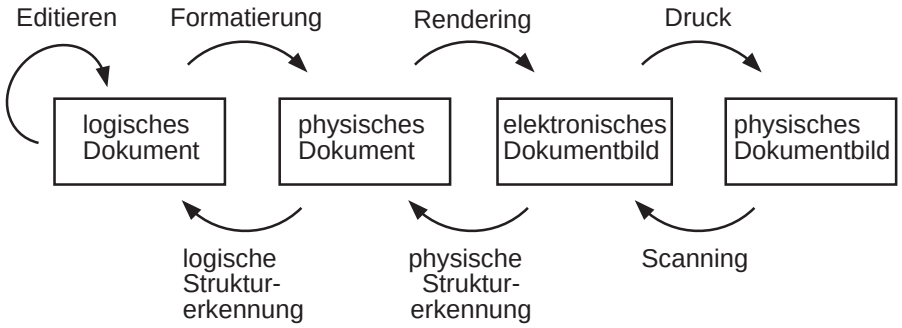


Abbildung 1.1: Produktion und Erkennung von Dokumenten.

digitalisiert und als Bild dargestellt. Aus dem Bild wird in der Folge das physische Dokument rekonstruiert. Folgende Verfahren gelangen dafür zur Anwendung: Segmentierungsalgorithmen detektieren Textblöcke, Zeilen und Wörter, isolieren Kopf- und Fusszeilen sowie Marginalnoten, und sie erkennen Regionen mit Bildern, Graphiken und anderen nicht-textuellen Elementen. Methoden der optischen Zeichenerkennung (OCR) klassifizieren anschliessend Buchstaben in einem Text und die optische Fonterkennung weist ihnen eine Schriftart zu. Aus diesen Informationen soll schliesslich das ursprüngliche logische Dokument rekonstruiert werden, durch Klassifikation und Gruppierung von physischen Elementen zu logischen Elementen. Man spricht in diesem Zusammenhang auch vom *Verstehen* von Dokumenten (engl. document understanding). Dieser Begriff ist insofern missverständlich, als er nichts mit dem Verstehen des Dokumentinhalts auf einer semantischen Ebene zu tun hat. Wir werden im weiteren daher den Begriff meiden und von Erkennung sprechen.

Die beiden letzten Stufen der Erkennung, insbesondere die der Rekonstruktion des logischen Dokuments, sind sehr komplex. Da es noch keine allgemein akzeptierten Methoden dafür gibt, sind diese Gegenstand aktueller Forschung.

1.1.2 Anwendungen

Anwendungen für die Dokumenterkennung finden sich in erster Linie in Bereichen, in denen die Verarbeitung von Dokumenten, oder allgemeiner die Verwaltung von Information einen wesentlichen Stellenwert einnimmt. In ihren Ursprüngen wurde die Dokumenterkennung in der Annahme entwickelt, dass längerfristig dem papierbasierten Dokument keine Zukunft beschieden sei. Die Anwendung der Dokumenterkennung sollte sich somit auf eine Übergangsfrist beschränken, während der sämtliche wichtigen Papierdokumente ein für alle-

mal digitalisiert würden. Alle neu erzeugten Dokumente wären gemäss dieser Annahme bereits in ihrer originalen, digitalen Form verfügbar, in der sie weiterverarbeitet oder archiviert würden.

Dem ist entgegenzuhalten, dass mit der Computerisierung der Informationsverarbeitung der Gebrauch von Papier eher zugenommen hat [Bun98]. Das papierlose Büro ist mehr denn je ein Mythos.

Doch selbst wenn man davon ausgeht, dass Dokumente künftig in ihrer originalen, digitalen Form verfügbar sind, bleibt die Dokumenterkennung ein wichtiges Hilfsmittel. Insbesondere in heterogenen Umgebungen werden regelmässig viele verschiedende Dokumentformate verwendet, je nach Verfügbarkeit von Programmen oder den vorgegebenen Standards, nach Typen von Dokumenten oder schlicht nach den Vorlieben der Anwender. Aufgrund der unzähligen Menge an existierenden Formaten gestaltet sich der Transfer von Dokumenten in andere Formate problematisch. Hier bietet sich das Dokumentbild als allgemeinstes Format an.

Neben seiner Allgemeinheit hat das Bildformat zudem den Vorteil, praktisch nicht zu veralten. Man darf davon ausgehen, dass Bilddokumente selbst nach Jahrhunderten für den Menschen interpretierbar bleiben[1]. Im Gegensatz dazu kann es bereits nach wenigen Jahren schwierig werden, ältere Versionen einer Textverarbeitungsdatei zu öffnen und zu bearbeiten.

Officeanwendungen

Das Büro ist eines der Haupteinsatzgebiete des Computers, für Private und Verwaltungen gleichermassen wie für Klein- und Grossbetriebe. Die potentiellen Anwendungen in diesem Bereich sind sehr vielfältig:

- Mit der Vernetzung von Computern gewinnt der Dokumentenaustausch innerhalb von Arbeitsgruppen immer mehr an Bedeutung. Wie bereits oben erwähnt, ist der Dokumententransfer von verschiedenen Plattformen oder Applikationen oft mit Informationsverlusten verbunden oder gar unmöglich, wenn kein direkter Konverter vom Quell- zum Zielformat existiert. Ein möglicher Ausweg ist der Transfer via das Bildformat, wie in Abbildung 1.2 gezeigt wird.

  Die Konversion des Dokumentes in ein Bildformat gehört zur Grundausstattung jeder Textverarbeitung. Die Aufgabe der Dokumenterkennung, die mit Back-End Konvertern ausgestattet die gewünschten Zielformate generiert, ist die Rückübersetzung in ein logisches Format. Die Anzahl

[1] Auch Jahrhunderte alte Schriften, wie mittelalterliche Bibeln, sind in ihrer Struktur und Schrift heute noch allgemein verständlich. Der semantische Gehalt hingegen ist nur Experten mit dem entsprechenden historischen und sprachlichen Wissen zugänglich.

der benötigten Konverter reduziert sich somit von $(n-1)^2$ beim Transfer mit direkten Konvertern auf n Back-End Konverter, sofern das Dokumentbild als einheitliches Basisformat gewählt wird.

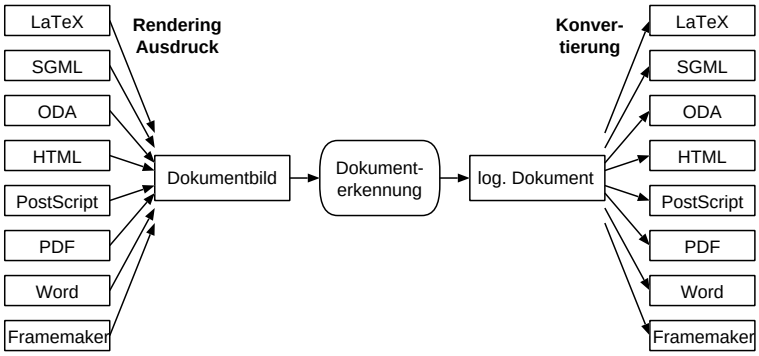


Abbildung 1.2: Konvertierung von Dokumentformaten mit Dokumentbild als einheitlichem Basisformat

- In der innerbetrieblichen Organisation spielen Systeme für Archivierung, Retrieval und Recherche von Dokumenten eine wichtige Rolle. Sie werden unter dem Begriff des *Dokument-Managements* zusammengefasst, das ein Teilbereich des *Workflow-Managements* ist. Die Dokumenterkennung kann hier für die Archivierung in einheitliche Formate, zur Unterstützung des Information-Retrieval und zur Vermeidung oder Linderung von Medienbrüchen eingesetzt werden [BAP98b, SP98, Lan98]. So ist beispielsweise ein System wünschbar, das eingehende Post und Faxe scannt, automatisch Sender- und Empfängeradresse detektiert und an die zuständige Stelle weiterleitet.

Archivierung

Die rasante Verbilligung der hochkapazitiven Speichermedien ermöglicht die wirtschaftliche Digitalisierung auch grosser, papierbasierter Archive. Die Vorteile dabei sind der schnellere, dezentrale Datenzugriff und ein verringerter Platzbedarf, die höhere Beständigkeit und beliebige Kopierbarkeit der Daten sowie niedrigere Kosten für Infrastruktur und Personal [SMBK95].

Mit Hilfe von Methoden des Information Retrieval sind zudem komplexere Abfragen als bei herkömmlichen Archiven möglich. Allerdings muss ein digitales Archiv indiziert werden, um einen optimalen Datenzugriff zu gewährleisten. Ei-

ne manuelle Indizierung kommt bei den typischerweise enorm grossen Datenbeständen von Bibliotheken, Verwaltungen und Grossbetrieben aus ökonomischen Gründen nicht in Frage. Hier hilft die Dokumenterkennung bei der automatischen Indizierung der Daten, oder sie kann die archivierten Daten aufbereiten und mit textueller und struktureller Information ergänzen [Doe97].

Telekom- und Postdienste

Trotz der immer grösseren Vernetzung von Computern haben die klassischen Kommunikationsdienste wie Telefon, Telefax und Briefpost ihre wichtige Bedeutung nicht verloren. Weil jedes Medium seine spezifischen Vorteile aufweist ist nicht zu erwarten, dass eines von ihnen in naher Zukunft verschwindet. Allerdings ist es bis heute kaum möglich, die Grenzen eines für die Übermittlung einer Nachricht gewählten Mediums zu überschreiten.

Hier bietet sich die Dokumenterkennung an, die durch Konversion und teilweise Interpretation von Nachrichten Medienbrüche transparent macht. So können zum Beispiel in Verbindung mit Sprachsynthesizern Briefpost-zu-Telefon Gateways eingerichtet werden; oder mit der Erkennung von Empfängeradressen auf Briefen wird es möglich, diese zu scannen und als Faxe weiter zu versenden. Das Ziel hierbei ist die Erreichung einer möglichst hohen Medientransparenz bei der Übermittlung beliebiger Nachrichten [Mil98].

Internet

Auch im Zusammenhang mit dem World Wide Web lassen sich eine ganze Reihe interessanter Anwendungen für die Dokumenterkennung finden [NSV97]. Serverseitig kann sie in Suchmaschinen eingesetzt werden, um eine Volltextsuche auf Bilder und PostScript-Dokumente zu erweitern, die sich im WWW oder in elektronischen Bibliotheken befinden [LZ98, ZL97]. In einer weiteren Stufe sind auch Abfragen denkbar, die die Struktur von Dokumenten berücksichtigen. Ein Beispiel hierfür ist die Suche nach einem Dokument unter Angabe des Autors und eines Keywords aus dem Abstract. Eine derartige Suchmethode, die die Struktur der durchsuchten Dokumente berücksichtigt, kann viel treffendere Resultate liefern als eine flache Volltextsuche.

Aber auch auf der Clientseite eröffnen sich Anwendungsgebiete. Obwohl HTML eigentlich für die logische Strukturierung von Dokumenten entworfen wurde, wird sie meistens als Layoutsprache missbraucht. Die eigentliche Struktur lässt sich in solchen Fällen nur aus der Präsentation, also dem Dokumentbild erschliessen. Werden für einen bestimmten Zweck wirklich strukturierte Webdokumente benötigt, so lässt sich das mit einer Dokumenterkennung am elegantesten bewerkstelligen.

Autorensysteme

In Textverarbeitungssystemen, die spezifisch streng strukturierte Dokumente erstellen, kann die Dokumenterkennung unterstützend eingesetzt werden. Im Vordergrund steht hierbei eine bisher noch nicht diskutierte Eigenschaft der Dokumenterkennung: die des *Lernens* eines Modells. Dokumentmodelle legen in Autorensystemen die Struktur von Dokumenten zwingend fest, was üblicherweise bei der Publikation unter Verlagen vorausgesetzt wird. Autoren sind jedoch im allgemeinen keine Computerexperten und werden daher von zu komplexen Modellen verwirrt. Es können nun aber diejenigen Dokumente, die Autoren eines bestimmten Sachgebietes erstellt haben, zum Modellernen herangezogen werden. Wenn die Autoren nur einen Teil des ursprünglichen Modells verwenden, wird das neu Gelernte eine Untermenge darstellen; genau diejenige, die für das bestimme Sachgebiet gebraucht wird [Aho96].

Alternativ dazu dienen Dokumente, die von Autoren mit herkömmlichen Textverarbeitungen erstellt wurden, als Basis zum Lernen eines Modells. Ein so gelerntes Modell kann nach einer allfälligen Nachbearbeitung durch Experten für die weitere Dokumentproduktion als verbindlich vorgeschrieben werden.

1.1.3 Ein ideales Dokumenterkennungssystem

Ein ideales oder universelles System erkennt und analysiert die ihm präsentierten Dokumente ohne weiteres Dazutun eines Operators fehlerfrei. Bei näherer Betrachtung wird jedoch klar, inwiefern verschiedene Anwendungen auch verschiedene Sichten von denselben Dokumenten haben, es also nicht eine „einzige korrekte“ Lösung des Erkennungsproblems gibt. Wahrscheinlich ist es auch auf lange Sicht hinaus nicht möglich, Dokumenterkennungssysteme so zu erweitern, dass sie fähig sind beliebig viele Schriftarten, Layouts und Strukturvarianten zu erkennen. Es wird vielmehr nötig sein, die Erkennungssysteme auf eingeschränkte Problemklassen zu spezialisieren.

Aus diesen Gründen ist es notwendig, die Systeme mit der Fähigkeit auszustatten, sich mit Hilfe des Benutzers an eine Problemklasse anzupassen, indem es von ihm lernt. Ein ideales System *interagiert* also mit dem Benutzer, *adaptiert* sich dabei an eine Problemklasse und erkennt *fehlertolerant* das Layout und die logische Struktur von Dokumenten.

1.1.4 Interaktivität

Wie bereits geschildert, ist es illusorisch, eine Dokumenterkennung vollautomatisch durchführen zu wollen. Die Systeme müssen vom Benutzer laufend angepasst und Fehler korrigiert werden. Aus ergonomischen Überlegungen darf sich die Rolle des Benutzers aber nicht darauf beschränken, die Erkennung zu

initialisieren, zu starten und schliesslich die Fehler zu korrigieren. Vielmehr sollte er in den Erkennungprozess eingebunden sein und ihn aktiv leiten und verfolgen. Beim Auftreten von Problemen konsultiert das System selbständig den Anwender, der aber auch seinerseits aktiv werden, und leitend eingreifen kann.

1.1.5 Adaption

Wird ein System mit einer neuartigen Problemklasse konfrontiert, ist deren Erkennung a priori unmöglich. Es ist auf den Benutzer angewiesen, der durch interaktive Präsentation von Beispielen und Prototypen eine *initiale Lernphase* einleitet. Das System generiert aus den gelieferten Informationen ein erstes, vorläufiges Modell für die Problemklasse. Vom System wird nach einer initialen Lernphase kein hundertprozentig korrektes Funktionieren erwartet. Es stösst auf neuartige und unbekannte Situationen und macht dabei Fehler. Bei jeder Fehlerkorrektur, die der Benutzer interaktiv durchführt, hat das System die Möglichkeit, den Grund für den Fehler zu eruieren und das Modell entsprechend zu verbessern. In solchen *inkrementellen Lernphasen* verfeinert sich das Modell und passt sich sukzessive an die Problemklasse an. Die Leistung des Systems nimmt folglich mit der Dauer seines Gebrauchs zu.

1.1.6 Fehlertoleranz

Es gibt mehrere Gründe für das Auftreten von Fehlern in der Dokumenterkennung. Die zugrundeliegenden Modelle können ungenügend sein (was gegebenenfalls Anlass zu einer inkrementellen Lernphase gibt), oder aber die zu erkennenden Dokumente sind verrauscht oder fehlerhaft. Die Ursachen dafür sind Verschmutzungen der Originale oder Diskretisierungsfehler beim Einscannen, von Autoren falsch gewählte Layouts oder Schriftarten sowie Tippfehler etc. Das System soll auf solche Probleme möglichst robust reagieren und allenfalls gewisse Abweichungen vom Modell tolerieren.

Ein System, das über eine einfache Fehlertoleranz verfügt, prüft kritisch die eigenen Resultate und informiert den Benutzer wenn Schwierigkeiten auftreten. Bei einer weitergehenden Fehlertoleranz vermag es selbständig Fehler zu korrigieren, indem es beispielsweise verschiedene, sich gegenseitig ergänzende oder ersetzende Erkennungsmethoden anwendet und deren Resultate gegeneinander abwägt.

1.2 Szenario

Das folgende Szenario soll die erwähnten Anforderungen verdeutlichen. Als Beispiel wird die Erkennung von Literaturhinweisen herangezogen. Dies mag für einen Forscher von Nutzen sein, der gewisse Literaturhinweise aus Publikationen in eine eigene Datenbank übernehmen möchte, ohne sie abzutippen. Er digitalisiert den Bibliographieabschnitt beispielsweise mit einem Handscanner. Er startet die Dokumenterkennung auf dem eingescannten Bild, selektiert gegebenenfalls einige Ausschnitte, deren Text und Struktur erkannt werden, und er speichert sie in einer Literaturdatenbank (z.B. BibTEX). Der Erkennungsvorgang soll nun etwas detaillierter betrachtet werden.

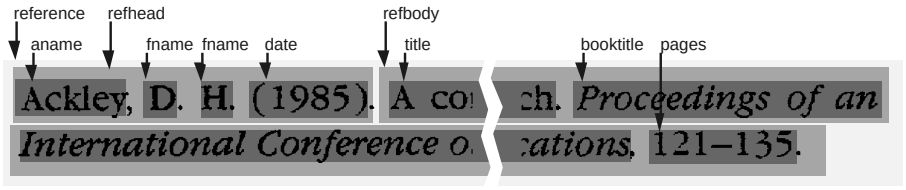


Abbildung 1.3: Manuell zum Eintrag hinzugefügte logische Labels.

In Abbildung 1.3 ist ein gescannter Literatureintrag dargestellt. Sofern das Erkennungssystem über kein Modell für Literatureinträge verfügt, wird eine initiale Lernphase eingeleitet. In dieser Phase definiert der Benutzer manuell die Struktur, indem er interaktiv Textbereiche auswählt und ihnen *logische Labels* assoziiert, wie Autor, Titel, Jahr und Verlag. Die Gesamtheit der logischen Labels und ihre typischerweise hierarchische Anordnung definiert die logische Struktur für einen spezifischen Eintrag. Mit der Extraktion der für das Modell relevanten Informationen schliesst das System die initiale Lernphase ab. Es ist nun befähigt, gleichartige Einträge selbständig zu erkennen.

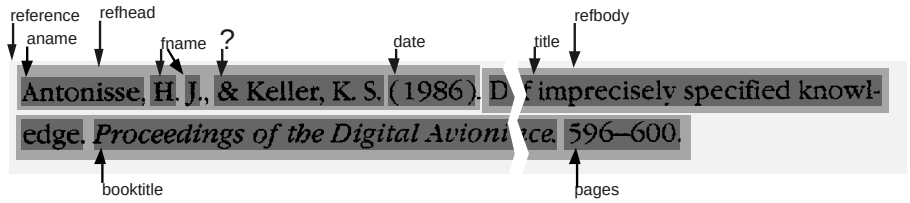


Abbildung 1.4: Die Erkennung schlägt fehl an einem neuartigen Eintrag.

Wenn das System im Laufe der Erkennung auf Einträge stösst, die mit den bis dahin im Modell gesammelten Informationen nicht analysiert werden können, wird der Benutzer konsultiert, wie in Abbildung 1.4 gezeigt. Das System vermag den Eintrag mit zwei Autoren nicht zu interpretieren, weil der gelernte Eintrag nur einen Autor aufführte. Daraus zieht es den Schluss, dass alle Einträge nur einen einzelnen Autor enthalten. Der Benutzer deblockiert die Situation durch manuelle Vervollständigung der Struktur, die anschliessend inkrementell gelernt wird. Das Modell beschreibt nun, dass Literatureinträge einen oder auch mehrere Autoren beinhalten können.

Nach dieser Betrachtung, was die Strukturerkennung dereinst idealerweise wird leisten können, folgt eine Beschreibung des für die vorliegende Arbeit gewählten Umfangs.

1.3 Forschungsgegenstand dieser Arbeit

Es wird noch Jahre dauern, bis Forschung und Entwicklung soweit gediehen sind, um annähernd so leistungsfähige Erkennungssysteme für Dokumente anbieten zu können, wie es der vorhergehende Abschnitt als wünschenswert darstellt. Die Forschung ist bis heute lediglich in der Lage, Teilaspekte der Dokumenterkennung zu behandlen und entsprechende Fragen zu beantworten. Bereiche wie OCR, Erkennung der physischen Struktur oder die Erkennung der logischen Struktur für spezifische Anwendungen sind isoliert beforscht worden, ohne in deren Integration zu einem einheitlichen System wesentliche Fortschritte gemacht zu haben. Funktionsfähigere und flexiblere Gesamtsysteme liegen in weiter Ferne.

Die hier präsentierte Arbeit beschränkt sich auf zwei Teilaspekte:

- Die Erforschung und Entwicklung einer Methode für die robuste *Erkennung* der logischen Struktur von Dokumenten.
- Die Erforschung und Entwicklung einer Methode, die ein initiales und inkrementelles, interaktives und benutzerassistiertes Lernen des Modells erlaubt, das für die Erkennung benötigt wird.

1.4 Übersicht und Aufbau dieser Arbeit

Die vorliegende Arbeit ist in 10 Kapitel unterteilt.

- Das folgende Kapitel präsentiert eine Systematik der Dokumentstrukturen und einige Standards für deren Repräsentation.

- Das Kapitel 3 gibt eine Übersicht der bisherigen Forschungsergebnisse auf dem Gebiet der Erkennung von Dokumentstrukturen und der Anpassung von Systemen an neuartige Dokumenttypen durch Lernen.
- Kapitel 4 führt ein statistisches Modell für Dokumentstrukturen ein und in Kapitel 5 wird gezeigt, wie initiales und inkrementelles Lernen darauf implementiert wird.
- In Kapitel 6 wird ein heuristischer Algorithmus sowie seine Implementierung in einem Prototyp vorgestellt. Er erkennt die logische Struktur auf gescannten und segmentierten Dokumenten.
- Die Tests, die mit dem Prototyp durchgeführt wurden und die dabei erzielten Resultate werden in Kapitel 7 aufgeführt.
- In Kapitel 8 werden Möglichkeiten der Integration des statistischen Modells in bestehende Standards besprochen.
- In Kapitel 9 werden mögliche Erweiterungen und Verbesserungen des vorgestellen Ansatzes diskutiert.
- Eine abschliessende Diskussion erfolgt in Kapitel 10.

Kapitel 2

Dokumentstrukturen

Laut *Websters New Collegiate Dictionary* ist ein Dokument

> *A material substance (as a coin or stone) having on it a representation of the thoughts of men by means of some conventional mark or symbol.*

Diese sehr allgemein gehaltene Definition steht im Gegensatz zur alltäglichen Vorstellung des Begriffs Dokument. Nach der handelt es sich um eine papierbasierte Repräsentation von Text, Bildern und Graphiken. Mit der digitalen Verarbeitung von Dokumenten ist es möglich, diese nicht mehr ausschliesslich auf Papier, sondern auch auf elektronischen Datenträgern darzustellen.

Die damit verbundene flexiblere Handhabung von Information führte zu einer bemerkenswerten Weiterentwicklung des Dokumentkonzeptes. So sind zur klassischen, sequentiellen Form, in der die Informationen im wesentlichen in einer eindimensionalen Leseordnung angeordnet sind, neue hinzugekommen.

- Bei *Hypertext* wird ein Dokument in kleinere, semantisch abgeschlossene Texteinheiten aufgesplittet. Diese werden mittels Verbindungen (sogenannten *Hyperlinks*) wieder zusammengefügt, jedoch nicht notwendigerweise in einer sequentiellen, sondern in jeder beliebigen Netzstruktur, entsprechend den semantischen Zusammenhängen der Texteinheiten.

- *Multimediadokumente* besitzen zusätzlich zur räumlichen eine zeitliche Dimension. Neben statischen Informationen wie Text und Bildern können sie Animationen, Sprache oder Videosequenzen enthalten. Ein Beispiel für diesen Typ ist eine Reparaturanleitung, die gewisse Handgriffe als Animation darstellt.

- *Dynamische Dokumente* verändern ihren Inhalt mit der Zeit. Interaktive Dokumente, eine Untergruppe der dynamischen Dokumente, sind in der Lage, auf Interventionen ihres Lesers in bestimmter Weise zu reagieren, indem sie beispielsweise Berechnungen durchführen oder kontextbezogene Inhalte anzeigen. Ein Beispiel dazu stellt eine interaktive Steuererklärung dar, die den Benutzer durch das Formular führt und selbständig Saldi etc. berechnet.

Es ist also unklar, wie der Begriff des Dokuments befriedigend definiert werden soll. Wenn eine Generalisierung des Begriffs durchaus Sinn macht, sollte er dennoch nicht überstrapaziert und von Begriffen wie *Benutzeroberfläche* und dergleichen deutlich abgegrenzt werden.

Wir werden uns auf die Betrachtung der statischen Dokumenttypen beschränken wie Zeitungsartikel, Bücher, Briefe und Karteikarten aber auch Hypertexte.

2.1 Das elektronische Dokument

Elektronische Dokumente nehmen im Verlauf ihrer Verarbeitung auf dem Computer verschiedene Formen an. Statische elektronische Dokumente, das heisst solche, die sich während der Betrachtungszeit nicht verändern, lassen sich adäquat in vier Abstraktionsstufen einteilen: das logische und das physische Dokument, den Textinhalt und das Dokumentbild.

2.1.1 Das logische Dokument

Die logische Struktur eines Dokuments beschreibt , wie der Inhalt organisiert ist, bevor er in einer spezifischen physischen Struktur dargestellt wird. Die logische Struktur zusammen mit dem Inhalt ergibt das *logische Dokument*. Das logische Dokument ist typischerweise die Sicht des Autors und Lesers, der den Text in logische Elemente gliedert. Einige Beispiele für logische Strukturelemente sind Sätze, Wörter und Zeichen, und auf höherem Niveau Absätze, Kapitel, Adressen und Fussnoten. Die gängigste Darstellung der logischen Struktur ist eine Baumstruktur: eine hierarchische Anordnung von logischen Elementen, wie Abbildung 2.1 zeigt. Das logische Dokument enthält von den hier diskutierten Repräsentationen am meisten Information und ist somit die abstrakteste.

Gebräuchliche Formate für der Darstellung sind ODA, LaTeX(mit Einschränkungen) und SGML, beziehungsweise dessen Instanzen HTML, QWERTZ, Linuxdoc etc.

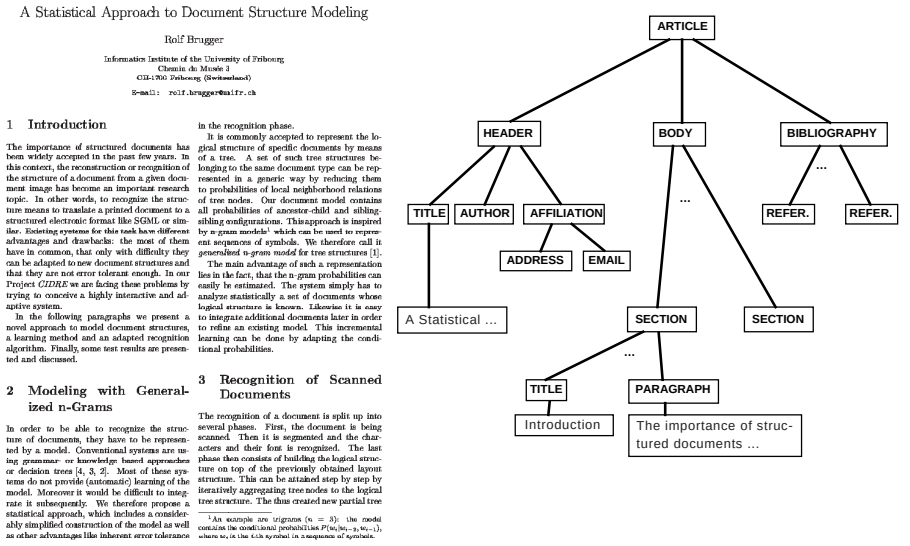


Abbildung 2.1: Die logische Sicht eines Dokuments.

2.1.2 Das physische Dokument

Die physische Struktur beschreibt, wie der Inhalt auf einem physischen Medium gesetzt wird. Die physische Struktur ergibt zusammen mit dem Inhalt das *physische Dokument*. Derselbe Inhalt lässt sich verschieden organisieren, was zu verschiedenen Layouts führt. Die Gestaltung wird bestimmt von Attributen der physischen Elemente wie Schriftart, Zeilenabstände, Zeilenausrichtung und Seitenabmessungen. Das physische Dokument ist die typische Sicht des Setzers oder Graphikers, der einen Text angemessen darstellt.

Die gebräuchlichsten Formate für die Darstellung in der Dokumentproduktion sind PostScript [Ado86], Portable Document Format (PDF) [Ado93] und DVI [Knu86]. Diese Standardformate sind optimiert für die Produktion von Dokumenten und daher für die Repräsentation der physischen Struktur in der Erkennung ungeeignet. Mit der Document Attribute Format Specification (DAFS) [RAF95] der Firma RAF Technology existiert nur ein spezialisiertes Format, das im Forschungsbereich der Dokumenterkennung zunehmende Bedeutung erlangt.

Ein weitreichender Konsens besteht darüber, welche Klassen von physischen Elementen nötig sind, um physische Strukturen für die Erkennung vollständig zu beschreiben [DDS+97]. Ein Dokument wird zunächst eingeteilt in *Seiten*. Diese enthalten Spalten oder *Blöcke*, welche sich weiter in *Zeilen*, *Wörter* und

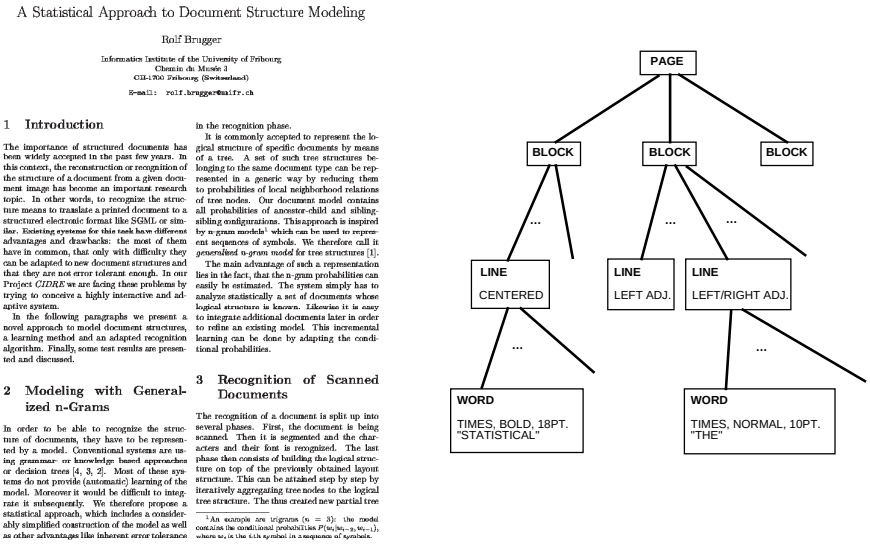


Abbildung 2.2: Die physische Sicht eines Dokuments.

Zeichen gliedern (Abbildung 2.2). Je nach System oder Standard werden Blöcke auch rekursiv verwendet.

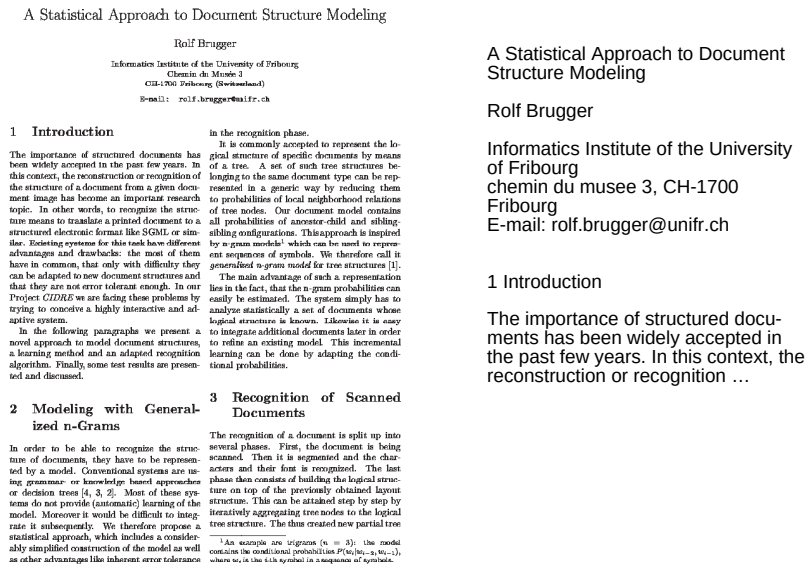
2.1.3 Der Dokumentinhalt

Der *Dokumentinhalt* bezeichnet die textuelle Information in einem Dokument. Auf unterster Ebene repräsentiert er eine Sequenz von Zeichen, die zu Wörtern, Sätzen und Absätzen zusammengefasst werden. Diese Elemente bilden eine einfache Struktur, die aus dem Zeichenstrom durch Interpretation von Separatorzeichen gebildet wird. Separatorzeichen sind Abstand, Zeilenumbruch und Satzzeichen. Der reine Inhalt ist nicht weiter strukturiert und stellt daher eine untere Abstraktionsebene dar.

Gebräuchliche Formate für die Darstellung sind ASCII- und UNICODE-Dateien.

2.1.4 Das Dokumentbild

Ein *Dokumentbild* ist eine zweidimensionale Anordnung von Pixeln beliebiger Farbe, im Fall von Dokumenten jedoch meist schwarz und weiss. Es wird durch



A Statistical Approach to Document Structure Modeling

Rolf Brugger

Informatics Institute of the University of Fribourg
Chemin du Musée 3
CH-1700 Fribourg (Switzerland)
E-mail: rolf.brugger@unifr.ch

1 Introduction

The importance of structured documents has been widely accepted in the past few years. In this context, the reconstruction or recognition of the structure of a document from a given document image has become an important research topic. In other words, to recognize the structure means to translate a printed document to a structured electronic format like SGML or similar. Existing systems for this task have different advantages and drawbacks: the most of them have in common, that only with difficulty they can be adapted to new document structures and that they are not error tolerant enough. In our Project *CIDRE* we are facing these problems by trying to conceive a highly interactive and adaptive system.

In the following paragraphs we present a novel approach to model document structures, a learning method and an adapted recognition algorithm. Finally, some test results are presented and discussed.

2 Modeling with Generalized n-Grams

In order to be able to recognize the structure of documents, they have to be represented by a model. Conventional systems are using grammar- or knowledge based approaches or decision trees [4, 3, 2]. Most of these systems do not provide (automatic) learning of the model. Moreover it would be difficult to integrate it subsequently. We therefore propose a statistical approach, which includes a considerably simplified construction of the model as well as other advantages like inherent error tolerance in the recognition phase.

It is commonly accepted to represent the logical structure of specific documents by means of a tree. A set of such tree structures belonging to the same document type can be represented in a generic way by reducing them to probabilities of local neighborhood relations of tree nodes. Our document model contains all probabilities of ancestor-child and sibling-sibling configurations. This approach is inspired by n-gram models[1] which can be used to represent sequences of symbols. We therefore call it *generalized n-gram model* for tree structures [1].

The main advantage of such a representation lies in the fact, that the n-gram probabilities can easily be estimated. The system simply has to analyze statistically a set of documents whose logical structure is known. Likewise it is easy to integrate additional documents later in order to refine an existing model. This incremental learning can be done by adapting the conditional probabilities.

3 Recognition of Scanned Documents

The recognition of a document is split up into several phases. First, the document is being scanned. Then it is segmented and the characters and their font is recognized. The last phase then consists of building the logical structure on top of the previously obtained layout structure. This can be attained step by step by iteratively aggregating tree nodes to the logical tree structure. The thus created new partial tree

[1] An example are trigrams ($n = 3$): the model contains the conditional probabilities $P(w_i|w_{i-2}, w_{i-1})$, where w_i is the i-th symbol in a sequence of symbols.

A Statistical Approach to Document Structure Modeling

Rolf Brugger

Informatics Institute of the University of Fribourg
chemin du musee 3, CH-1700
Fribourg
E-mail: rolf.brugger@unifr.ch

1 Introduction

The importance of structured documents has been widely accepted in the past few years. In this context, the reconstruction or recognition ...

Abbildung 2.3: Der textuelle Inhalt eines Dokuments.

Rastern aus dem physischen Dokument generiert. Ein Dokumentbild kann ohne weitere Verarbeitung direkt durch ein Rasterausgabegerät wie Drucker oder Bildschirm wiedergegeben werden. Das wiedergegebene Bild enthält keinerlei direkte Strukturinformation und stellt deshalb aus informationstechnischer Sicht die niedrigste Abstraktionsstufe von Dokumenten dar[1].

Grundsätzlich lässt sich ein Dokumentbild in jedem Format darstellen. Besonders eignen sich auf Text spezialisierte, verlustfrei komprimierende Bildformate, wie CCITT Fax Group 3/4 enconding des TIFF Standards (Tagged Image File Format) [Ald92].

[1] Für den Menschen hingegen, der ein Dokument lesen will, ist das Dokumentbild die bevorzugte Darstellung. (Für die die auditive oder taktile Wahrnehmung werden wiederum andere Darstellungen bevorzugt.) Er kann aus der visuellen Wahrnehmung des Dokumentbildes am schnellsten und einfachsten dessen Struktur erfassen. Dies erklärt auch die Beliebtheit von what-you-see-is-what-you-get (WYSIWYG) Textverarbeitungen, die während des Editierens eines Dokumentes sein Bild fortwährend berechnen und am Bildschirm anzeigen.

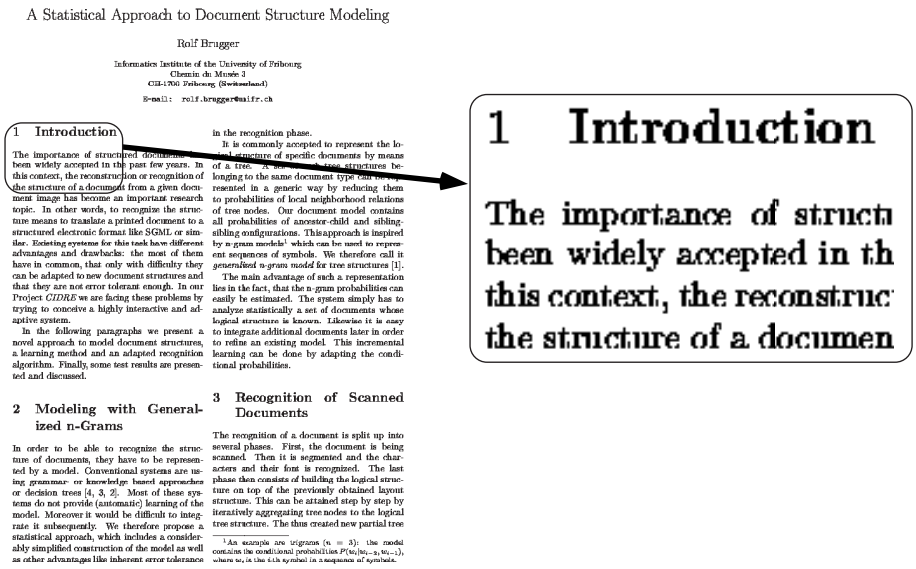


Abbildung 2.4: Das Dokument in Bildform.

2.2 Die logisch-physische Differenzierung

Im vorhergehenden Abschnitt wurde gezeigt, was eine Dokumentstruktur ist und wie sie dargestellt wird. Wir betrachten nun genauer, warum eine Strukturierung sinnvoll ist, und richten dabei das Augenmerk besonders auf die logische und physische Struktur.

Das Editieren eines Dokuments ohne jeglichen Begriff einer Struktur, ist wie das Programmieren in Assembler. Ähnliche Parallelen zur strukturierten Programmierung zieht Reid [Rei89] (siehe Tabelle 2.1). Die Strukturierung gibt dem Autor eine bessere Übersicht. Er kann leichter Dokumentteile wiederverwenden und die Gefahr, im Aufbau und in der Darstellung Fehler zu begehen, wird reduziert.

Ein Dokument lässt sich am leichtesten auf der logischen Ebene manipulieren. Die interessantesten Anwendungen, die in der Einleitung besprochen wurden, benötigen die logische Struktur. Formatkonversionen lassen sich effizient und leicht auf (logische) SGML Dokumente anwenden, mit der *transformation language*, die als Teil von DSSSL spezifiziert ist [ISO96] (SGML und andere Standards werden im Abschnitt 2.4 eingehend besprochen). Information Retrieval Aufgaben beschränken sich auf eine Kombination von Volltextsuche und

	Programm	**Dokument**
Beispiele für Strukturelemente	Variablennamen, Blockstruktur, Typen, Kommentare, Einrückungen	Blockstruktur, Bezeichner (logische Tags)
editierbare / modifizierbare Form	Sourcecode	logisches Dokument
nicht reversibler Prozess	Compilation	Formatierung
ausführbare / interpretierbare Form	Maschinencode	physisches Dokument
was wird ausgedrückt	Computation	Inhalt
Beispiele für Linking / late Binding	Sprungadressen, Bibliotheksvarianten, Optimierungen	Auflösung, Qualität, Papierformat, Fontzuweisung

Tabelle 2.1: Analogie zwischen strukturierter Programmierung und strukturierten Dokumenten nach Reid [Rei89]

Parsing des logischen Dokuments, zum Beispiel mit dem frei verfügbaren SGML-Parser sp [Cla99]. SGML Dokumente lassen sich problemlos direkt archivieren. Die Archivierung der logischen Repräsentation bietet zudem den Vorteil, sehr bescheidene Anforderungen an die Speicherkapazität eines Archivs zu stellen, und dies obwohl sie die Repräsentation ist, die am meisten Information enthält.

Wenn ein Dokument nicht mehr verändert werden soll, wird es formatiert und in ein physisches Dokument übergeführt. Die meisten Entscheidungen, die die Darstellung betreffen, werden beim Formatieren getroffen. Der Vorteil des formatierten Dokuments liegt darin, dass für einen Ausdruck kein aufwendiger und kostspieliger Formatierer mehr benötigt wird, und auf diese Weise auch eine plattformunabhängige Ausgabe ermöglicht. Ein weiterer Vorteil liegt in der einfacheren Handhabung. Physische Dokumente bestehen in der Regel aus einer Datei, während logische sehr viele Dateien umfassen können. So hat sich für den Download langer und qualitativ hochstehender Dokumente das physische Format PDF und seltener PostScript eingebürgert.

2.2.1 Zusammenhänge zwischen physischer und logischer Struktur

In der vorgestellten Systematik hat die duale, physische und logische Sichtweise von demselben Dokument nur dann einen Sinn, wenn zwischen ihnen auch Verbindungen bestehen. Es ist allerdings nicht trivial, jene Verbindungen systematisch sichtbar machen zu wollen.

Am einfachsten ist ein Bezug in den Extremitäten auszumachen. Zwischen Zeichen im logischen und Zeichen im physischen Dokument besteht eine 1:1 Beziehung; ebenso auf der Wurzelebene, die in beiden Fällen das gesamte Dokument repräsentiert. Einzige Ausnahmen stellen gewisse graphische Objekte, sowie Kopf- und Fusszeilen dar.

Für Elemente zwischen den Extremen existieren im allgemeinen keine eindeutigen Entsprechungen. So besteht ein Absatz aus mehreren Zeilen und ist selbst Teil einer oder auch mehrerer Textspalten. Eine systematische Definitionsmöglichkeit einer Beziehung ist, jedes logische Element als aus n physischen Elementen zusammengesetzt zu betrachten. Im Interesse einer hohen Abstraktion sollten grösstmögliche physische Elemente zur Konstruktion herangezogen werden. Demnach besteht ein Absatz aus n Zeilen, ein Autorenname aus n Wörtern und eine Jahreszahl aus 2 oder 4 Zeichen. Die physische Struktur wird an der logischen gleichermassen aufgehängt. Da sich dies für die Realisierung der Dokumenterkennung als praktisch erwiesen hat, wird es in Kapitel 4 ausführlicher behandelt.

2.2.2 Mikro- und Makrostrukturen

Im Zusammenhang mit der Dokumenterkennung wird zwischen Systemen für die Erkennung von Mikrostrukturen und solchen zur Erkennung von Makrostrukturen unterschieden. Diese Unterscheidung ergibt sich, weil aufgrund der Verschiedenheit der Problematik Erkennungssysteme unterschiedlich konzipiert werden mussten. Die Erkennung der Makrostruktur hat zum Ziel, die logische Struktur von der Wurzel bis auf das Niveau von Absätzen zu konstruieren. Hingegen konstruiert die Erkennung der Mikrostruktur die logische Struktur innerhalb von Absätzen oder ähnlich grossen logischen Elementen. Dabei wird zum Beispiel ein Datum mit seinen Bestandteilen wie Tag, Monat, Jahr erkannt oder ein Literaturhinweis mit Autor, Titel, Verlag und Erscheinungsjahr.

2.3 Die spezifisch-generische Differenzierung

Textintensive Dokumente können in eine grosse Zahl von Typen unterteilt werden (Lehrbücher, Romane, Magazine, Zeitungsartikel, Geschäftsbriefe etc.). Je-

der der Typen hat seine charakteristische Art, *welche* logischen Elemente vorkommen, *wie* sie zusammengefügt werden und wie ihr Layout beschaffen ist.

Die Charakteristika einer Klasse von Dokumenten werden in einem *Modell* definiert. Demnach ist ein logisches Dokument eine Instanz des logischen Modells, und analog ist ein physisches Dokument eine Instanz des physischen Modells. Abbildung 2.5 verdeutlicht den Zusammenhang zwischen logischer und physischer Struktur einerseits und Modell und Instanz andererseits.

	Modell	**Instanz**
logisch	generisch logisches Dokument	spezifisch logisches Dokument
physisch	generisch physisches Dokument	spezifisch physisches Dokument

Abbildung 2.5: Die logisch-physische und generisch-spezifische Dimension in Dokumentstrukturen.

Das *logische Modell*, oder die generische logische Struktur, besteht aus einer Aufzählung, der für die Darstellung eines Dokuments erlaubten logischen Elemente zusammen mit den Regeln, in welcher Weise sie anzuordnen sind. So müssen beispielsweise in der Dokumentklasse „Buch“ Kapitel stets mit einem Titel beginnen, und Literaturhinweise dürfen sich nicht irgendwo sondern nur am Schluss eines Kapitels befinden.

Das *physische Modell*, oder die generische physische Struktur, ist eine Beschreibung, wie die logische Struktur in die physische übersetzt wird, und beschreibt implizit oder explizit das Layout einer Dokumentklasse.

Die Separierung von Modell und Instanz hat eine Reihe praktischer Vorteile für den Produktionsablauf. Verwenden mehrere Autoren ein gleiches logisches Modell, so ist garantiert, dass ihre Dokumente eine einheitliche logische Struktur besitzen. Dies erleichtert erheblich die Weiterverarbeitung, wie die Transformation in andere Dokumenttypen, die einheitliche Formatierung oder die Ablage in Datenbanken.

Durch die Verwendung eines getrennten Modells für das Layout ist es andererseits möglich, das Layout einer Klasse von Dokumenten einheitlich zu gestalten, und es zu ändern, ohne die originalen, logischen Dokumente anzupassen.

In der Dokumenterkennung haben Modelle eine Bedeutung bei Systemen, die nicht für eine einzelne Dokumentklasse konzipiert wurden. Solche Systeme lassen sich flexibel für die Erkennung verschiedener Dokumentklassen einset-

zen, wenn entsprechende Erkennungsmodelle zur Verfügung stehen. Leider sind keine einfachen Methoden bekannt, mit denen sich Erkennungsmodelle aus den entsprechenden Produktionsmodellen automatisch generieren lassen. Als Alternative wird in dieser Arbeit eine Methode vorgeschlagen, mit welcher sich ein Erkennungmodell unter der Mithilfe eines Benutzers erlernen lässt.

2.4 Standards für Dokumentmodelle

Im Zuge der Vereinheitlichung und Vereinfachung des Dokumentenaustausches wurden Standards für Dokumentmodelle geschaffen. Es haben sich im Laufe der Zeit zwei Standards durchgesetzt: SGML und ODA. Während SGML die grössere Verbreitung erfahren hat, scheint sich ODA nicht auf breiter Ebene durchsetzen zu können.

2.4.1 SGML

Die Standard Generalized Markup Language (SGML) [ISO86b] definiert einen Standard für die Repräsentation von Dokumentstrukturen, der hauptsächlich zur Darstellung von logischen Dokumenten entwickelt wurde. Das Prinzip besteht darin, dass der textuelle Inhalt mit *Markup* angereichert und so logisch strukturiert wird. Unter Markup versteht man definierte Symbole, die vom Inhalt syntaktisch eindeutig unterscheidbar sind. Konkret werden SGML Dokumente als ASCII Datei dargestellt; Abbildung 2.6 zeigt Markupsymbole durch in spitze Klammern eingefasste Bezeichner.

Die Art und Weise, wie logische Elemente angeordnet sind, wird in einer Document Type Definition (DTD) festgelegt. Die DTD repräsentiert das logische Modell und stellt im wesentlichen eine kontextfreie Grammatik dar. Die wichtigsten Operatoren sind im Beispiel in Abbildung 2.6 aufgeführt und haben die folgenden Bedeutungen: “,“: Sequenz, “|“: Alternative, “*“: Iteration, “+“: Iteration mindestens einmal, “?“: Option.

Erst vor wenigen Jahren wurde mit Document Style Semantics and Specification Language (DSSSL) [ISO96] ein Standard zur Spezifikation von physischen Modellen für SGML-Modelle verabschiedet. In einer *Style Specification* wird die physische Präsentation von logischen Elementen festgelegt und damit das Layout des gesamtem Dokuments. Da eine statische Deklaration für die komplexen Formatierungsoperationen nicht genügend expressiv ist, wurde in DSSSL mit einem LISP-Dialekt eine vollständige Programmiersprache integriert.

HTML, eine Instanz von SGML, hat als Sprache des World-Wide-Web in jüngerer Zeit zu einer explosionsartigen Verbreitung des Internet beigetragen. Um den Bedürfnissen bezüglich der Publikation von Online-Dokumenten besser Rechnung zu tragen, wurde die Metasprache SGML wesentlich vereinfacht und

SGML-Dokument (Instanz)

```
<article>
<title>SGML - Quo vadis?</title>
...
<chap>
  <chaptit> Vorwort </chaptit>
  <par>
    Mit der zunehmenden
    Bedeutung
    ...
  </par>
</chap>
</article>
```

DTD (Modell)

```
<!element article - -
          title, header, abstract,
          chap+, biblio? >

<!element chap    - -
          chaptit,
          (par* | subchap*) >
...
```

Abbildung 2.6: Ausschnitte aus einem SGML-Dokument und der dazugehörigen DTD.

daraus XML entwickelt. Es ist zu erwarten, dass XML in naher Zukunft eine erhebliche Bedeutung im WWW erlangen wird.

2.4.2 ODA

Die Open Document Architecture (ODA) [ISO86a] umfasst ein Dokumentmodell für die Spezifikation der logischen und physischen Struktur. Sie verwendet ein binäres Format zur Repräsentation der Struktur sowie des Inhalts. Wie aus Abbildung 2.7 ersichtlich ist, werden die logische und physische Struktur als zwei Bäume dargestellt, die gemeinsame, identische Blätter besitzen, welche Fragmente des Inhalts darstellen. Auf diese Weise bringt ODA die 1:1 Beziehung der logischen und physischen Struktur auf der Inhaltsebene elegant zum Ausdruck.

Das Dokumentmodell wird ebenfalls als Baum dargestellt, der zusätzlich mit Attributen versehen ist. Ein erster Satz von Attributen dient der Definition der Beziehung zwischen logischen Objekten. Hier einige Beispiele: “SEQ“ Sequenz, “REQ“ obligatorisch, “OPT“ optional, “CHO“ alternativ, “REP“ repetitiv. Ein zweiter Satz von Attributen definiert die Darstellung des Inhalts, wie die zu verwendenden Fonts, Zeilenausrichtungen und Umbruchregeln etc.

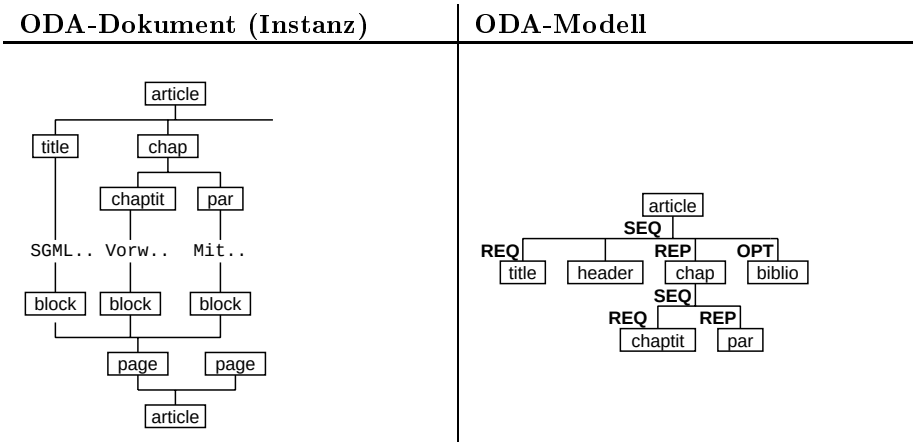


Abbildung 2.7: Ausschnitte aus einem ODA-Dokument und dem dazugehörigen Modell.

2.5 Zusammenfassung

Das Konzept des Dokuments wurde mit der automatisierten Informationsverarbeitung auf den Computer übertragbar und dadurch erheblich erweitert. Ein Dokument kann interaktive Elemente enthalten und auf komplexe Weise mit anderen Dokumenten verknüpft sein.

Vernünftigerweise muss sich die Erkennung zur Zeit auf textintensive Dokumente beschränken, die zudem in ihrer Struktur einem gewissen Modell gehorchen.

Abbildung 2.8 zeigt im Überblick die Phasen der Produktion und Erkennung von Dokumenten, die dabei verarbeiteten Daten sowie häufig verwendete Formate.

Ein Autor bearbeitet ein Dokument in seiner logischen Repräsentation. Die Struktur des logischen Dokuments ist durch ein logisches Modell eingeschränkt. Sie bildet eine Instanz des Modells. Durch Formatierung wird das physische Dokument erzeugt, dessen Layout weitgehend vom physischen Produktionsmodell bestimmt wird. Daraus lässt sich plattformunabhängig ein Dokumentbild generieren, welches druckbar und vom Menschen leicht lesbar ist. Mittels der physischen Dokumenterkennung kann aus einem Dokumentbild ein physisches Dokument erzeugt werden, das sich aber im Aufbau von einem physischen Dokument für die Produktion grundlegend unterscheidet. Der Grund dafür liegt in der Tatsache, dass Produktion und Erkennung komplett verschiedene Prozesse

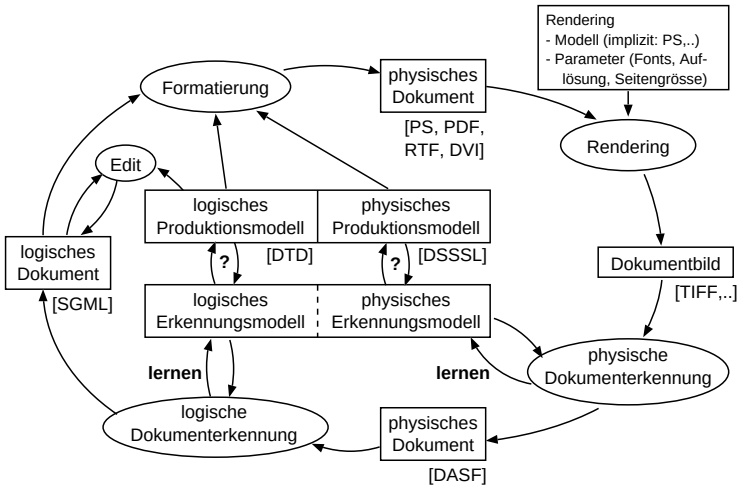


Abbildung 2.8: Die Zusammenhänge zwischen Dokumentmodellen und dem Produktions- und Erkennungsprozess.

darstellen und infolgedessen in Form von Modellen unterschiedlich parametrisiert sein müssen.

In einer letzten Stufe wird das logische Dokument rekonstruiert. Modelle, die alle benötigten Parameter sowie Strukturinformationen enthalten steuern die Erkennungsprozesse. Durch Austausch der Modelle lässt sich die Erkennung auf verschiedene Dokumentklassen anwenden. Es sind bisher kaum Methoden bekannt, mit denen sich Erkennungs- und Produktionsmodelle ineinander überführen lassen. In der Dokumenterkennung existieren ferner, mit Ausnahme von DAFS, keine Standardformate für Dokumentmodelle und -instanzen.

Kapitel 3

Dokumenterkennung

In den Anfängen der Dokumenterkennung, die bis in die 60er Jahre zurückgehen, interessierten sich Forscher und Anwender vorrangig für die optische Zeichenerkennung (OCR). Obgleich die sich damit ergebende Problematik nicht trivial ist, hat die optische Zeichenerkennung bis heute einen hohen Reifegrad erreicht. Auch existieren mittlerweile eine Reihe kommerzieller Systeme, die teilweise genügend preiswert sind, um von einem breiten Publikum genutzt zu werden.

Eine weitergehende Extraktion von Information aus Dokumenten wurde erst mit der Verfügbarkeit erschwinglicher und leistungsfähiger Rechner zu Beginn der 80er Jahre möglich. Dies führte zu einer wahren Explosion von Forschungsaktivitäten rund um die Dokumenterkennung. Deren Themenbereiche umfassen Handschriftenerkennung und Verifikation von Unterschriften, Erkennung nicht lateinischer Alphabete und Fonterkennung, Extraktion der physischen und logischen Struktur sowie die Erkennung von Tabellen, technischen Zeichnungen und Plänen, mathematischen Formeln und Musiknoten.

Wenn wir daraus die Entwicklung der Strukturerkennung herausgreifen und näher betrachten, so stellen wir erhebliche Fortschritte fest. Erste Systeme waren auf enge Problembereiche spezialisiert und konnten grundsätzlich nicht auf neue Bereiche angepasst werden. Mit der Einführung von Modellen in der Dokumentproduktion wurde deren Konzept auf den Erkennungsprozess angewandt. Durch Verändern des Modells liess sich ein Erkennungssystem flexibel an neue Problemklassen adaptieren. Da die Erstellung von Erkennungsmodellen eine komplexe und zudem fehleranfällige Arbeit darstellt, die überdies nur von Experten zu bewerkstelligen ist, konzentriert sich die aktuelle Entwicklung auf das automatisierte Lernen von Modellen. In diesem noch jungen Forschungsbereich existieren bislang nur eine Handvoll neuerer Publikationen [OK95, HT98, ICD95, ICD97, BW97].

Die folgenden Abschnitte geben einen Überblick über die wichtigsten wissenschaftlichen Publikationen, die in den letzten zwei Jahrzehnten erschienen sind. Wir beschränken uns dabei auf Themen, die insofern in einem direkten Bezug zu dieser Arbeit stehen, als sie die die Erkennung der logischen Dokumentstruktur und das Lernen von Erkennungsmodellen behandeln.

3.1 Bisherige Forschungsarbeiten

Es ist nicht einfach die diversen Dokumenterkennungssysteme einander direkt gegenüberzustellen. Jedes System hat seine spezifischen Charakteristika auf einer breiten Skala von Eigenschaften. Da ein möglichst systematischer Vergleich zwischen den Systemen angestrebt wird, erfolgt eine kurze Auflistung der zu vergleichenden Eigenschaften:

Genereller Ansatz: Welche, aus anderen Bereichen der Informatik bekannten Methoden der Mustererkennung oder Schlussfolgerung wird verwendet? Man stösst auf syntaktische Ansätze und auf Ansätze der Artificial Intelligence im weitesten Sinne. Rein statistische Ansätze sind bisher nicht dokumentiert.

Lernen / Erkennen: Handelt es sich um ein reines Erkennungssystem, oder bietet es Möglichkeiten, sich durch Lernen an besondere Situationen anzupassen?

Strukturtyp: Erkennt das System nur die physische oder auch die logische Struktur?

Generizität: Wie flexibel kann das System an neue Problembereiche angepasst werden? Ist das System „hart programmiert“ oder basiert es auf einem manuell erstellten oder einem gelernten Modell?

Modellrepräsentation: Wie wird das Modell repräsentiert? Man findet attribuierte und nicht attribuierte Grammatiken, Baumgrammatiken, Entscheidungsbäume, Frames, semantische Netzwerke.

Modellinformationen: Welche Informationen werden im Modell dargestellt? Beispiele sind geometrisches Layout, Fontinformation, logische Struktur, Listen mit Schlüsselwörtern und Wörterbücher.

Methode der Analyse: Auf welche Art erfolgt die Analyse? Top-down, bottom-up, mixed mode Analyse oder gerichtete Erkennung?

Die folgenden Abschnitte sind Kurzbeschreibungen der wichtigsten Arbeiten aus dem Bereich der logischen Strukturerkennung von Dokumenten sowie dem Lernen von logischen Dokumentmodellen. Eine stichwortartige Übersicht der Erkennungsmethoden ist zudem in Tabelle 3.1 gegeben.

3.1.1 Wissensbasierte Ansätze

InfoPortLab [Bay93, BBMS94] wurde von Thomas Bayer und seinen Mitarbeitern bei Daimler-Benz Research entwickelt. Ihr Ziel ist die Extraktion von logischen Elementen aus Geschäftsbriefen unter Verwendung einer wissensbasierten Architektur.

Das Modell wird durch ein semantisches Netz in der Sprache FRESCO repräsentiert. Es enthält einerseits statisches, deklaratives Wissen über den logischen und physischen Aufbau (Dokumentmodell) und andererseits dynamisches, prozedurales Wissen über anzuwendende Algorithmen und Erkennungsstrategien. Das Dokumentmodell besteht aus einer Hierarchie von logischen Konzepten, die untereinander mit „has-a"-Relationen verbunden sind (z.B Address has-a Street). Zudem sind sie über „subclass"-Relationen mit physischen Konzepten verknüpft, wodurch sie alles Wissen von ihnen erben. Im algorithmischen Modell wird jeder Algorithmus als Konzept mit drei Attributen beschrieben: Vorbedingungen, Nachbedingungen und Berechnungsaufwand. Mit Hilfe dieser Informationen optimiert das System selbständig den Ablauf von Algorithmen. Der Ansatz beinhaltet limitierte Lernmöglichkeiten, zum Beispiel für die Anpassung interner fuzzy-Parameter.

Der Inferenzalgorithmus läuft in drei Phasen ab: Erstens werden Hypothesen für Konzepte gebildet (top-down), zweitens werden die physischen Elemente aufbereitet (bottom-up) und schliesslich werden die logischen Konzepte interpretiert und instanziert (bottom-up).

In der ersten Testphase wurden Betragsfelder in Einzahlungsscheinen sowie Daten in Briefen erkannt. In der zweiten Testphase [BW95] wurden Sender- und Empfängeradresse, Datum und Begrüssung aus 161 Geschäftsbriefen extrahiert.

OfficeMAID [Den92, DBH+94] wurde in der Arbeitsgruppe von Andreas Dengel am DFKI in Kaiserslautern entwickelt. Es ist für die Analyse, Interpretation und automatische Weiterleitung von Geschäftsbriefen konzipiert. Auf der zuvor extrahierten Hierarchie von physischen Objekten wird die logische Struktur konstruiert. Dies läuft in drei Stufen ab; Als erstes werden Hypothesen für die Orte und Anordnung von logischen Objekten generiert. Das dabei zugrundeliegende Modell ist ein geometrischer Entscheidungsbaum, GTree. Die Hypothesen werden anschliessend mit 108 Regeln aus einer statistischen Datenbank für Geschäftsbriefe in ihrer lokalen Kohärenz bewertet. Schliesslich erfolgt

eine letzte Plausibilitätsprüfung durch einen Vergleich der Hypothesen mit 18 globalen Konsistenzregeln. Die Methode erlaubt ein initiales und inkrementelles Lernen des GTree [Den93]. Die Konsistenzregeln sind handgeneriert, basierend auf statistischen Daten.

Der Ansatz wurde an 100 einseitigen Geschäftsbriefen getestet, wobei in 91% der Dokumente die korrekte logische Struktur gefunden wurde.

PLRS [EMS95] ist eine Entwicklung der Arbeitsgruppe um Floriana Esposito der Universität von Bari. Ihr Ziel ist ein System für die automatische Klassifikation und die Erkennung der logischen Struktur von Dokumenten. In einer globalen top-down Analyse werden Spalten, Textblöcke, Abbildungen und Tabellen detektiert. Die daran anschliessende bottom-up Analyse gruppiert Blöcke zu logischen Elementen zusammen. Beide Analysestufen werden von einem aus PROLOG-Regeln bestehendem Modell gesteuert. Die Regeln werden in einer Lernphase automatisch generiert [EMS93].

Der Prototyp wurde an 200 einseitigen Dokumenten aus den Bereichen Geschäftsbriefe, Faxcovers, Inhaltsverzeichnisse und einzelne Seiten aus Konferenzproceedings getestet.

Graphein [Che93] entwickelte Yannick Chenevoy am CRIN-CNRS in Nancy für die Erkennung einer breiten Palette von Dokumenttypen. Als Basis verwendet er eine Blackboard-Architektur, über die Spezialisten Informationen austauschen. Hypothesen werden in einer mixed-mode (top-down und bottom-up) Strategie erzeugt und mit einem Entropie-Score bewertet. Das Modell orientiert sich an ODA und ist in ATOME, einem LISP-Dialekt, formalisiert. Der Ansatz wurde mit der Erkennung von Karteikarten eines Bibliothekkatalogs validiert.

Später wurde dieser Ansatz von Tunde Akindele aufgegriffen und mit Mechanismen für das automatische Lernen des Modells erweitert [Aki95b, Aki95a]. In ein initiales Modell, das die physische Struktur eines Dokuments repräsentiert, werden schrittweise weitere Dokumente der gleichen Klasse hineinfusioniert. Mit Standardmethoden aus der Theorie der Baumgrammatiken wird das Modell nach Fusionsschritten regelmässig vereinfacht.

DeLoS [NS95] wurde von Debashish Niyogi in der Gruppe von Sargur Srihari (CEDAR) an der Universität von Buffalo entwickelt. Dabei handelt es sich um ein wissensbasiertes System für die Extraktion der logischen Struktur aus Dokumentbildern. PROLOG-Klauseln repräsentieren Kontrollregeln für die lokale Steuerung des Inferenzprozesses, Strategieregeln für die globale Steuerung der Erkennungsstrategie und Wissen über die Interpretation des Layouts. Die Regeln implementieren eine gemischte Top-down Bottom-up Kontrollarchitektur. Durch Erweiterung der Regeln, kann das System an neue Problemklassen

adaptiert werden. Die Tests wurden an 44 einseitigen Dokumentbildern aus Tageszeitungen durchgeführt.

IDA [Kre93] wurde von Joachim Kreich bei der Siemens AG in München mit dem Ziel entwickelt, die logische Struktur von Briefköpfen zu erkennen. Im Modell wird zwischen Domain Knowledge (Modell der Dokumentstruktur) und Control Knowledge (Prozesswissen, wie die Domain Knowledge gehandhabt wird) unterschieden. Die erzeugten Hypothesen werden auf dem Dokumentmodell basierend in ihrer Konsistenz geprüft und bewertet. Das Ergebnis wird in einer best-first Suche aus den generierten Hypothesen ermittelt. Es wurden nach unseren Kenntnissen keine Testresultate publiziert.

IDUS [TL96] entwickelten Suzanne Taylor und Mark Lipshutz bei Lockheed Martin in Paoli. Das Ziel ihres Projekts ist die Dokumentkonversion und die Extraktion von logischen Elementen aus Geschäftsbriefen. Ein hybrides Modell dient zur Analyse der logischen Objekte (funktionale Objekte in der Terminologie von Taylor/Lipshutz). Intra-Objekt-Wissen ist in Fuzzy-Logik implementiert, Inter-Objekt-Wissen mit Standard PROLOG-Prädikaten. Die Tests wurden an 120 einseitigen Geschäftsbriefen durchgeführt.

3.1.2 Syntaktische Ansätze

OSCAR-II [Hu94, Ing88, Ing91] wurde von Tao Hu in der Gruppe von Rolf Ingold an der Universität Fribourg entwickelt. Das Ziel dieses Projekts ist die Erkennung der logischen Struktur von Dokumenten aus verschiedenen Klassen. Das Erkennungsmodell besteht aus einer attribuierten, kontextfreien Grammatik. Die generisch logische Dokumentstruktur wird durch die kontextfreien Produktionsregeln repräsentiert, während die physische Präsentation der logischen Elemente in den entsprechenden Attributen dargestellt ist. Die logische Struktur eines Dokuments wird beim Parsen seiner physischen Struktur unter Verwendung einer top-down Strategie extrahiert. Durch den Einsatz von Fuzzy-Logik Matching und dynamischen, heuristischen Schwellwerten wird eine Fehlertoleranz in der Erkennung erreicht. Die Komplexität der Erkennung lässt sich mit einer dynamic-programming Suchstrategie erheblich reduzieren.

Die Tests wurden durchgeführt an einem einseitigen Dokument, an einem 16-seitigen Gesetzestext und an einem 22-seitigen Kapitel aus einem Chemielehrbuch.

TWIG [Con93] ist ein Projekt, das von Alan Conway am Hitachi Laboratory in Dublin unterhalten wird. Das Ziel ist die Konversion von Papierdokumenten in eine logische, strukturierte Form (SGML). Nach der Segmentierung

und Zeichenerkennung wird die Sequenz von physischen Elementen mit einem von einer Page-Layout-Grammar getriebenen Parser analysiert. Das Resultat ist eine Sequenz von logischen Elementen. Durch nochmaliges Parsen mit einer Logical-Structure-Grammar wird die logische Struktur erzeugt. Eine Page-Layout-Grammar ist eine kontextfreie Grammatik, die leicht erweitert wurde, um einfache geometrische Beziehungen zwischen Elementen darstellen zu können. Es werden keine Resultate präsentiert.

3.1.3 Lernen von Modellen

MarkItUp! [FX93] von Peter Fankhauser und Yi Xu wurde am GMD in Darmstadt entwickelt. Das Ziel ist die automatisierte Generierung von DTD-Grammatiken aus gegebenen SGML-Dokumenten. Das System hat die Fähigkeit, aus einzelnen, von Hand strukturierten SGML-Dokumenten eine DTD zu konstruieren. Wenn mit der gelernten DTD bei einem späteren Parsen von unstrukturierten Dokumenten Fehler auftreten, so kann der Benutzer diese mit einem Strukturediteor korrigieren und die somit angepassten Dokumente inkrementell lernen lassen.

Das System kann nur auf Dokumente angewendet werden, die auf der Zeichenebene hoch strukturiert sind, da keinerlei Layoutinformation berücksichtigt wird. Dies ist zum Beispiel bei Literaturverzeichnissen der Fall, mit denen auch die Tests durchgeführt wurden. Die erzeugten DTDs sind kryptisch und manuell nur sehr schwer nachzubearbeiten.

„System Ahonen“ [Aho96] wurde entwickelt von Helena Ahonen an der Universität von Helsinki. Ihr Ziel ist die Entwicklung eines Werkzeugs für die automatische Generierung einer DTD aus vorgegebenen SGML-Dokumenten. Spezielle Vorkehrungen werden getroffen, um Mehrdeutigkeiten einer DTD zu beheben (SGML-DTD's müssen *1-unambiguous* sein). Es ist ein vorrangiges Ziel des Projekts, DTD's zu erzeugen, die vom Menschen verstanden und nachbearbeitet werden können.

FRED [Sha95] wurde von Keith Shafer am Online Computer Library Center (OCLG) in Dublin/Ohio entwickelt. Das Ziel ist die automatische Generierung einer DTD für Dokumente, die zwar SGML-strukturiert sind, zu denen aber keine DTD mehr existiert. Die Generierung ist so konzipiert, dass sie als Batch-Prozess ablaufen kann.

System	**Autoren**	**Ansatz**	**Repräsentation des Modells**	**Testdaten**	**Besonderheiten**
InfoPortLab	Bayer	Semantisches Netz	Semantisches Netz mit Concepts, Attributes, Descriptions, Constraints	161 einseitige Geschäftsbriefe	Fehlertolerant, parallele Architektur
OfficeMAID	Dengel	Knowledge based	geometrisches Layout (GTree), lokale und globale Konsistenzregeln	100 einseitige Geschäftsbriefe	GTree kann initial und inkrementell gelernt werden
PLRS	Esposito	Knowledge based	PROLOG-Regeln	200 einseitige Dokumente aus diversen Klassen	Ermöglicht initiales Lernen der PROLOG-Regeln
Graphein	Chenevoy, Belaïd	Multi-Agent Blackboard	ODA-Ähnliche Hierarchie von log. Objekten. Blätter haben phys. Attribute	—	Intelligentes Hypothesenmanagement, Lernalgorithmus in neuerer Publikation vorgeschlagen
DeLoS	Niyogi, Srihari	Knowledge based	PROLOG-Regeln: Knowledge-, Control-, Strategy-Regeln. 160 Regeln (erweiterbar)	44 einseitige Seiten aus Tageszeitungen	Mixed mode Analyse
IDUS	Taylor, Lipshutz	Feature classification, Knowledge based	Fuzzy features, PROLOG-Prädikate	—	Blackboard Kontrollarchitektur, Matching von Schlüsselwörtern, multiple knowledge sources
OSCAR-II	Ingold, Hu	syntaktisch	attribuierte Grammatik	mehrseitige Dokumente aus verschiedenen Klassen	Fehlertoleranz (fuzzy-logic matching), dynamic-programming Suchstrategie
TWIG	Conway	syntaktisch	Page-Layout-Grammar, Logical-Structure-Grammar	—	erzeugt direkt SGML-Dokumente

Tabelle 3.1: Übersicht der diskutierten Ansätze für die Erkennung der logischen Struktur

3.2 Das Projekt *CIDRE*

Nachdem sich an unserem Institut eine noch junge Arbeitsgruppe für Dokumenterkennung etabliert hatte, wurde 1994 ein neues Projekt ins Leben gerufen. Die Motivation für das Projekt *CIDRE*[1] (cooperative and interactive document reverse engineering) war eine grundlegende Neubeurteilung der Problematik der Dokumenterkennung, in dessen Verlauf eine Reihe von Publikationen erschienen sind [BBI95, BBZI96b, BBZI96a, BK97, BZI93, BZI97, BBI98, BI98a, RI98]. Im folgenden werden die drei Hauptaspekte des Projekts erläutert: Interaktivität, kooperative Systemarchitektur und lernfähige Modelle. Die beiden ersten Aspekte bilden das Schwergewicht in der Dissertation von Fédéric Bapst [Bap98a], während der letzte Gegenstand der vorliegenden Arbeit ist.

Der Anspruch auf eine vollautomatische Dokumenterkennung wird in *CIDRE* aufgegeben. Diese Entscheidung gründet auf der Einsicht, dass mit den heute verfügbaren Mitteln der Informatik der Ertrag am optimalsten ist, wenn ein Dokumenterkennungssystem von einem Anwender assistiert wird. Dies erfordert eine hohe Interaktivität zwischen Anwender und System. Die Rolle des Anwenders darf sich aber nicht auf die eintönige Arbeit beschränken, die Fehler des Systems zu korrigieren. Vielmehr soll das System die Interaktionen nutzen, um die eigene Leistung fortwährend zu verbessern und somit langfristig auf immer weniger Eingriffe des Anwenders angewiesen zu sein.

Ein weiterer Schlüssel für die Verbesserung der Dokumenterkennungsleistung liegt in der Systemarchitektur. Die klassische Architektur ist eine Aneinanderreihung von einzelnen Prozessen. *CIDRE* bricht mit diesem Paradigma, unter dem Postulat, dass eine Sequentialisierung der Verarbeitung für die Dokumenterkennung ungeeignet ist. Als Alternative wird eine Parallelisierung vorgeschlagen, die es erlaubt, dass verschiedene Prozesse gleichzeitig an einem Problem arbeiten. Durch Kooperation ergänzen sie sich gegenseitig und handeln selbständig Verarbeitungsabläufe aus.

Schliesslich wird dem Erkennungsmodell eine wichtige Rolle zugesprochen. Wenn in klassischen Systemen das Modell als ein Parameter der Erkennung betrachtet wird, so ist es in *CIDRE* eher ein Resultat davon. Da in typischen Fällen kein Erkennungsmodell verfügbar ist, wird dessen Erstellung in den gesamten Erkennungsprozess eingebunden. Mit inkrementellen Lernfähigkeiten ausgestattet, verfeinert es sich mit jeder Interaktion des Anwenders, um schliesslich eine Dokumentklasse möglichst vollständig zu modellieren. Somit ist es sogar denkbar, ein weitgehend verfeinertes Erkennungsmodell in ein Standard-Produktionsmodell umzuformen.

[1] Das Projekt *CIDRE* wurde unterstützt vom Schweizerischen Nationalfonds zur Förderung der wissenschaftlichen Forschung; Beitrag Nr. 21-42'355.94.

3.3 Zusammenfassung

Die optische Dokumenterkennung ist ein intensiv beforschter Bereich. Das gilt insbesondere für die Zeichenerkennung (OCR) und grundlegende Techniken der physischen Strukturerkennung (Segmentierung). Die Erkennung der logischen Dokumentstruktur findet in jüngerer Zeit zunehmend Beachtung. Aufgrund der hohen Komplexität des Erkennungsproblems kommen Ansätze aus der Forschung der Artificial Intelligence (AI) zur Anwendung: semantische Netzwerke, in LISP oder PROLOG implementierte Reasoning Systeme und Fuzzy-Logik.

Die vorgestellten Systeme wurden zumeist mit relativ wenigen und dazu einseitigen Dokumenten aus nur einer Klasse getestet. Innerhalb eines solcherart eingeschränkten Problembereichs lassen sich zufriedenstellende Resultate erzielen. Es scheint jedoch, dass komplexere und längere Dokumente derzeit nicht bearbeitet werden können. Daher ist ein Trend hin zu hybriden Multi-Expert Systemen zu beobachten, in denen unabhängige Module gemeinsam und kooperativ Erkennungsprobleme zu lösen versuchen.

Der Aspekt der benutzerassistierten Adaption eines Systems an eine neue Problemklasse wird selten in Betracht gezogen. Ein Grund dafür mag sein, dass weitgehend unbekannt ist, wie komplexe Modelle der AI (semantische Netzwerke, PROLOG-Programme etc.) automatisch gelernt werden können. Ein weiterer Grund ist vermutlich, dass die Systeme für ein korrektes und fehlertolerantes Funktionieren auf Handoptimierungen in ihren Modellen angewiesen sind.

Eine bemerkenswerte Tatsache ist ferner, dass kaum ein Erkennungssystem Fontinformationen verarbeiten kann. Dies ist nur damit zu erklären, dass die optische Fonterkennung (OFR) erstaunlicherweise lange Jahre völlig vernachlässigt wurde.

Kapitel 4

Ein statistisches Dokumentmodell

In den vergangenen drei Dekaden ist ein neuer Forschungszweig in der Informatik entstanden, der allgemein grosses Interesse und rege Forschungsaktivitäten weckt: Die *künstliche Intelligenz* (AI, artificial intelligence). Dieser sehr weitläufige Begriff umfasst eine Reihe von Anwendungen, von denen einige Beispiele aufgeführt sind:

- Diagnose, Vorhersage und Entscheidungen in komplexen Situationen mit Hilfe von Expertensystemen, zum Beispiel Diagnose einer Krankheit aufgrund beobachteter Symptome
- Interpretation und Klassifikation komplexer Objekte, zum Beispiel Identifikation von Personen in Bildern (Computer Vision) oder Buchstabenerkennung (OCR)
- Komplexes Problemlösen, zum Beispiel Schachspielen
- Verarbeitung natürlicher Sprache, zum Beispiel für das Diktieren von Texten in Systemen mit automatischer Spracheingabe
- Automatisches Beweisen von Theoremen

In der AI-Forschung wurden Methoden entwickelt, die sich grob in die folgenden Familien einteilen lassen: regelbasierte Methoden der formalen Logik (first order logic, non-monotonic logic), Methoden der formalen Sprachen (Parsing mit klassischen Grammatiken, Baumgrammatiken, etc.), Reasoning Systeme (semantische Netzwerke). Eine weitere wichtige Familie zur Lösung der oben

genannten Problembereiche bilden die statistischen Ansätze [Han93]. Es ist allgemein üblich, Ansätze dann als statistisch zu bezeichnen, wenn darin verwendete Methoden oder Repräsentationen in einem Zusammenhang zur Wahrscheinlichkeitsrechnung stehen. Beispiele für statistische Ansätze sind:

Bayessche Klassifikation ist eine grundlegende und dennoch leistungsfähige Methode der Klassifizierung von Vektoren.

Belief Networks beschreiben die Verknüpfungen von Zufallsvariablen entsprechend ihrer Abhängigkeiten untereinander. In Bayesschen Netzwerken entspricht eine Verbindung einer bedingten Wahrscheinlichkeit. Sie drücken Zusammenhänge aus wie beispielsweise „Bei Glukose im Urin besteht zu 70% die Wahrscheinlichkeit einer Diabeteserkrankung.“. Belief Networks werden bevorzugt in Expertensystemen eingesetzt.

Entscheidungsbäume sind stochastisch, wenn die Pfade mit Unsicherheitsfaktoren (Wahrscheinlichkeiten) attribuiert sind. Auch sie werden in Expertensystemen eingesetzt.

Markov-Modelle beschreiben Ketten von Ereignissen mit bedingten Wahrscheinlichkeiten (n-Gramme). Auf sie wird anschliessend näher eingegangen, weil sie am nächsten mit dem später vorgeschlagenen statistischen Dokumentmodell verwandt sind. Markov-Modelle werden erfolgreich in der Erkennung von Handschriften und gesprochener Sprache eingesetzt.

Probabilistische Grammatiken sind eine Erweiterung klassischer Grammatiken, bei denen Produktionsregeln mit der Wahrscheinlichkeit ihrer Anwendung attribuiert sind. Sie werden für die Modellierung der Syntax natürlicher Sprachen verwendet.

Fuzzy-Logik ist kein echter statistischer Ansatz. Die Darstellung von Plausibilitäten linguistischer Variablen hat aber grosse Ähnlichkeiten mit Wahrscheinlichkeitsverteilungen. Fuzzy-Logik wird unter anderem in komplexen Steuerungsprozessen eingesetzt.

Neuronale Netze wie Multilayer Perzeptron, Kohonen Map oder Hopfield Netze verwenden in der Regel statistikähnliche Methoden für das Lernen interner Gewichte. Neuronale Netzwerke werden für die Klassifizierung eingesetzt.

Statistische Methoden verbinden eine Reihe von Vorteilen, die sie von anderen Ansätzen abheben. Eine erste Stärke ist die leichte Lernmöglichkeit ihrer Modelle. Im Falle der Bayesschen Netzwerke oder der Fuzzy-Logik erfolgt das Lernen durch Beobachtung oder Befragung von Experten; eine weitergehende

Aufbereitung der Daten ist nicht nötig. Markov-Modelle und Neuronale Netze passen sich sogar selbständig an ein Problem an. Im weiteren erleichtern statistische Ansätze die Verarbeitung von unsicherer Information, da sie tolerant auf Unschärfen in den zu verarbeitenden Daten reagieren. Schliesslich gelangen statistische Ansätze dort zur Anwendung, wo nicht notwendigerweise eine perfekte Lösung, sondern eine Annäherung daran gesucht wird.

Um von den Vorteilen der leichten Lernbarkeit und Fehlertoleranz für die Dokumenterkennung einen Nutzen ziehen zu können, wurden die genannten Ansätze einer detaillierten Untersuchung unterzogen. In dieser Evaluation der statistischen und nicht-statistischen Ansätze haben sich n-Gramm-Modelle als besonders vielversprechend herausgestellt. Sie werden in den folgenden Abschnitten näher betrachtet.

4.1 n-Gramm Modelle

Im Folgenden wird der Begriff der Markov-Kette eingeführt und das statistische Modell der n-Gramme definiert. n-Gramme werden insbesondere im Zusammenhang mit der Verarbeitung natürlicher Sprachen verwendet [Cha93], wo sie Ereignissequenzen (zum Beispiel Buchstaben, Laute oder Wörter) modellieren. Anschliessend wird gezeigt, wie sich das Konzept der n-Gramm-Modelle erweitern lässt, um komplexere Strukturen wie Bäume oder Graphen beschreiben zu können.

4.1.1 Klassische n-Gramme

Gegeben sei ein System, das sich jeweils in einem von m Zuständen $S_1, \ldots, S_m$ befindet. Zu regelmässigen, diskreten Zeitpunkten vollzieht das System eine Zustandsänderung gemäss Übergangswahrscheinlichkeiten, die den Zuständen zugeordnet sind. Es seien $t = 1, 2, \ldots$ die Zeitpunkte der Zustandsänderungen und q_t der aktuelle Zustand zum Zeitpunkt t. Eine vollständige, probabilistische Beschreibung des Systems beschreibt die Wahrscheinlichkeit des Systemzustandes zum Zeitpunkt t, wenn alle vorangegangenen Zustände bekannt sind, mit:

$$P(q_t = S_{i_0} \mid q_{t-1} = S_{i_1}, q_{t-2} = S_{i_2}, \ldots, q_1 = S_{i_t}),$$

mit $1 \leq i_k \leq m$. Üblicherweise wird das System aber dahingehend vereinfacht, dass lediglich die $n-1$ vorangehenden Zustände berücksichtigt werden:

$$P(q_t = S_{i_0} \mid q_{t-1} = S_{i_1}, q_{t-2} = S_{i_2}, \ldots, q_{t-n+1} = S_{i_{n-1}}).$$

Ein solches probabilistisches System wird Markov-Kette $(n-1)$-ter Ordnung genannt. Die Menge aller bedingten Wahrscheinlichkeiten $P(q_t = S_i \mid$

$q_{t-1} = S_j, \ldots, q_{t-n+1} = S_l$) heisst n-Gramm-Modell für die Sequenz von Zuständen [Rab90]. Häufig benutzte Werte für n sind $n = 2$ und $n = 3$, wobei die entsprechenden Modelle Bigramm- beziehungsweise Trigramm-Modell genannt werden.

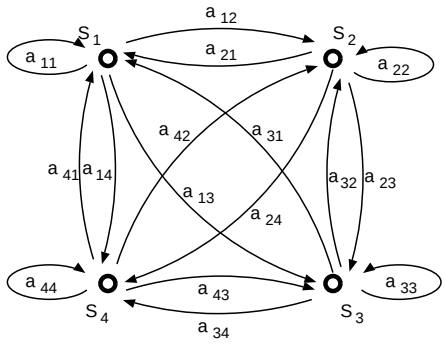


Abbildung 4.1: Markov-Kette erster Ordnung mit vier Zuständen.

Wie Abbildung 4.1 zeigt, sind die Koeffizienten a_{ij} die Übergangswahrscheinlichkeiten vom Zustand S_i zum Zustand S_j. Sie bilden insgesamt ein Bigramm-Modell:

$$a_{ij} = P(q_t = S_j \mid q_{t-1} = S_i).$$

Da die Koeffizienten a_{ij} per Definition stochastischen Bedingungen genügen, gilt:

$$a_{ij} \geq 0$$

$$\sum_{j=0}^{m} a_{ij} = 1.$$

Konkret lassen sich n-Gramm-Modelle auf zwei Arten anwenden: Für die Vorhersage von Zuständen sowie für die Messung der Wahrscheinlichkeit von Zustandsketten.

Bei der *Vorhersage* geht man davon aus, dass eine Kette von Zuständen $\ldots, q_{t-2}, q_{t-1}$ gegeben und der nachfolgende Zustand q_t zu bestimmen sei. Die Wahrscheinlichkeit, dass sich q_t im Zustand S_k befindet, ist im Fall eines Trigramm-Modells:

$$a_{ijk} = P(q_t = S_k \mid q_{t-1} = S_j, q_{t-2} = S_i).$$

Der am ehesten auftretende Zustand ist S_{i^*} mit

$$i^* = \max \arg_{1 \leq k \leq m}(a_{ijk}).$$

Die zweite Möglichkeit der Anwendung von n-Gramm-Modellen ist die *Messung* der Wahrscheinlichkeit von gegebenen Zustandssequenzen. Es seien eine endliche Kette von Zuständen $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$ sowie ein n-Gramm-Modell (im Beispiel hier ein Bigramm-Modell) gegeben. Die a priori Wahrscheinlichkeit der Zustandskette ist gegeben durch:

$$P(S_{i_1}, S_{i_2}, \ldots, S_{i_t}) = a_{i_1} \cdot a_{i_1 i_2} \cdot a_{i_2 i_3} \cdot \ldots \cdot a_{i_{t-1} i_t}$$

Zum obigen Ausdruck ist anzumerken, dass der erste Koeffizient a_{i_1} als einziger kein Bigramm darstellt, sondern die a priori Wahrscheinlichkeit des Auftretens des Startzustandes.

4.1.2 Anwendung von n-Grammen

Als Anwendungsbeispiele für n-Gramme ist die Nachkorrektur in OCR-Systemen zu nennen. Einfache OCR-Systeme isolieren Buchstaben und identifizieren sie unabhängig voneinander. Dabei treten als typische Fehler Verwechslungen in visuell ähnlichen Buchstabengruppen auf, wie beispielweise ri↔n, rn↔m oder l↔I. Unter Einbezug des Kontexts können solche Fehler unter Umständen wieder rückgängig gemacht werden, wenn auf einem n-Gramm-Modell basierend verschiedene Alternativen gegeneinander abgewogen und die wahrscheinlichste gewählt wird. So werden die im folgenden gezeigten Korrekturen möglich:

vvenn	→	wenn
zusamrnen	→	zusammen
weiI	→	weil

nicht jedoch:

Hirnbeere	→	Himbeere

4.1.3 Hidden Markov Modelle

Wie wir im vorangehenden Abschnitt gesehen haben, bietet ein n-Gramm-Modell eine einfache Möglichkeit für die Modellierung von sequentiellen Strukturen. Aufgrund ihrer Einfachheit eignen sich n-Gramm-Modelle allerdings nur dort, wo sehr einfache Strukturen repräsentiert werden müssen, oder wo in der Anwendung ein hoher Unsicherheitsfaktor toleriert wird (beispielsweise in der Nachkorrektur von OCR-Resultaten).

Eine wesentliche Erweiterung des n-Gramm-Ansatzes stellen die Hidden-Markov-Modelle (HMM) dar. Ein HMM besteht aus den folgenden fünf Komponenten [Rab90, Cha93]:

1. Die N Zustände $S = \{S_1, \ldots, S_N\}$. Der Zustand zum Zeitpunkt t ist q_t.
2. Die M Beobachtungen (Alphabet), repräsentiert durch die Symbole $V = \{V_1, \ldots, V_M\}$. Die Beobachtung zum Zeitpunkt t ist O_t.
3. Die Übergangswahrscheinlichkeiten der Zustände $A = \{a_{ij}\}$,
$$a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i), \ i \leq j \leq N.$$
4. Die Auftretenswahrscheinlichkeit von Beobachtungen $B = \{b_j(k)\}$,
$$b_j(k) = P(V_k \text{ bei } t \mid q_t = S_j), \ i \leq j \leq N, \ i \leq k \leq M.$$
5. Die Wahrscheinlichkeiten der Initialzustände $\pi = \{\pi_i\}$,
$$\pi_i = P(q_1 = S_i), \ i \leq j \leq N.$$

Ein vollständiges HMM ist somit $\lambda = (S, V, A, B, \pi)$, wobei M und N implizit darin enthalten sind. Abbildung 4.2 zeigt den Aufbau eines HMM. Das System befindet sich zu jedem diskreten Zeitpunkt t in einem Zustand q_t, der die Werte $S_1 \ldots S_N$ annehmen kann. Im darauf folgenden Zeitabschnitt $t+1$ erfährt das System eine Zustandsänderung von q_t nach q_{t+1}. Der Zustandsübergang wird mit den Bigramm-Wahrscheinlichkeiten $a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i)$ modelliert. Jedem Zustand S_i ist eine *Beobachtung* V_k zugeordnet, die wiederum mit der diskreten Bigramm-Wahrscheinlichkeitsverteilung $b_j(k) = P(V_k \text{ bei } t \mid q_t = S_j)$ modelliert ist.

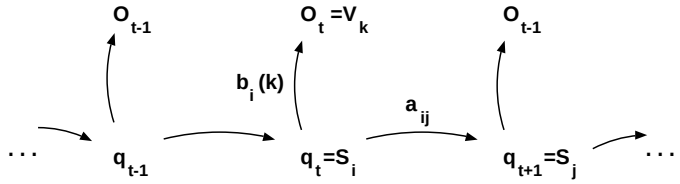


Abbildung 4.2: Ausschnitt einer HMM-Kette.

Ein HMM λ lässt sich nun anwenden, um die Wahrscheinlichkeit einer Beobachtungssequenz $O = O_1 O_2 \ldots O_T$, also $P(O \mid \lambda)$ zu berechnen. Der Evaluationsalgorithmus des HMM arrangiert dabei selbständig die „unter“ den Beobachtungen liegenden Zustände in der Weise, dass $P(O \mid \lambda)$ maximal wird.

Mehrere HMM können somit für die Klassifizierung von Beobachtungssequenzen verwendet werden. Dazu wird für jede Klasse C_v ein HMM λ^v erstellt. Die Klasse einer Beobachtungssequenz entspricht dann dem HMM, das $P(O \mid \lambda^v)$ maximiert:

$$v^* = \max \arg_{\forall v}(P(O \mid \lambda^v)).$$

Ein wichtiger Vorteil von HMMs ist ihre Lernfähigkeit und ihre Unempfindlichkeit gegenüber der Länge von Beobachtungssequenzen sowie Verschiebungen von Teilen innerhalb der Beobachtungssequenzen. Diese Eigenschaften sind bei OCR von Handschrift und in der Erkennung gesprochener Sprache willkommen, weshalb HMMs in diesen Bereichen mit Erfolg eingesetzt werden. Wenn zum Beispiel die Beobachtungen O_i Phonemen entsprechen, so soll ein HMM als Modell für ein ganzes Wort auch dann ansprechen, wenn einzelne Phoneme des Wortes kürzer oder länger artikuliert werden. Alternative Klassifizierungssysteme, wie neuronale Netze, sind in solchen Fällen weniger geeignet.

Trotz ihrer Leistungsfähigkeit hat sich keine Einsatzmöglichkeit der HMMs für unser Problem der Dokumentmodellierung ergeben. Die Gründe dafür sind, dass einerseits inkrementelle Lernschritte nur sehr umständlich implementiert werden können, und dass sie andererseits für die Konstruktion der logischen Struktur nicht anwendbar sind, wie sich später in Kapitel 6 zeigen wird.

Der folgende Abschnitt führt nun eine alternative Erweiterung der n-Gramm-Modelle ein, die sich für die Modellierung von Dokumentstrukturen besser eignet als HMMs.

4.1.4 Generalisierte n-Gramme

Bisher wurde gezeigt, wie sich Ereignissequenzen mit n-Grammen modellieren lassen. Nun stellt sich die Frage, ob auch komplexere Strukturen mit n-Grammen beschrieben werden können. Dass dies möglich ist, soll an einen Beispiel illustriert werden, in dem ein Gitter (Matrix) von Zuständen modelliert wird.

Gegeben sei ein Gitter mit 3×3 Elementen, das jedes den Zustand $S_0 = 0$ oder $S_1 = 1$ einnimmt. Wir definieren nun zwei Gruppen von Bigrammen:

$$r_{ij} = P(x = S_j \mid R(S_i, x))$$

$$t_{ij} = P(x = S_j \mid T(S_i, x)),$$

wobei $R(S_i, x)$ (beziehungsweise $T(S_i, x)$) bezeichnet, dass das Element x der rechte beziehungsweise obere Nachbar von S_i ist. Analog zur Modellierung von Sequenzen von Elementen repräsentieren die n-Gramme also Beziehungen zwischen benachbarten Elementen.

1	1	1		1	1	1		0	1	1		0	0	1
1	1	1		0	1	1		0	0	1		0	0	0
0	1	1		0	0	1		0	0	0		0	0	0

Abbildung 4.3: Die vier möglichen Muster, die der Strukturdefinition entsprechen.

Wir definieren nun eine Struktur im 3×3 Gitter. Es sei das Gitter durch eine von oben links nach unten rechts verlaufende Diagonale in zwei Hälften geteilt, wobei die untere linke Nullen und die obere rechte Einsen enthält (siehe Abbildung 4.3). Eine solche Struktur ist durch ein Modell M_{RT} definiert, das die folgenden Bigramme enthält:

$$r_{01} = \frac{2}{5},\ r_{00} = \frac{3}{5},\ r_{11} = 1,\ r_{10} = 0$$

$$t_{01} = \frac{2}{5},\ t_{00} = \frac{3}{5},\ t_{11} = 1,\ t_{10} = 0.$$

Die angegebenen Bigramm-Wahrscheinlichkeiten entsprechen dem theoretischen Wert, wenn die vier in Abbildung 4.3 gezeigten Muster gleich häufig vorkommen. Auf welche Weise n-Gramm-Wahrscheinlichkeiten numerisch zu berechnen sind, wird später in Kapitel 5 über das Lernen von Modellen gezeigt.

Grundsätzlich könnte die Modellierung der Gitterstruktur auch mit anderen Relationen als R und T realisiert werden. Denkbar wäre ein Modell M_{XY} mit Bigrammen, die den Bezug eines Elementes zu einem benachbarten Element in der Diagonalen repräsentieren:

$$\begin{aligned} M_{XY} &= (x_{01}, x_{11}, x_{10}, x_{00}, y_{01}, y_{11}, y_{10}, y_{00}) \\ x_{ij} &= P(x = S_j \mid X(S_i = x)) \\ y_{ij} &= P(x = S_j \mid Y(S_i = x)), \end{aligned}$$

wobei $X(S_i, x)$ (beziehungsweise $Y(S_i, x)$) bezeichnet, dass das Element x der linke obere, beziehungsweise rechte obere Nachbar von S_i ist. Zudem müssen nicht notwendigerweise Beziehungen zwischen direkt benachbarten Elementen berücksichtigt werden. Welche Elementbeziehungen eine Struktur optimal modellieren, hängt vom konkreten Problem ab und muss in empirischen oder theoretischen Studien erforscht werden. So kann die Messung der Entropie von n-Grammen Aussagen über die Modellqualität machen. Die Entropie von n-Grammen wird im Kapitel 5 über das Lernen von Modellen eingeführt und besprochen.

Nun fragt sich, ob generalisierte n-Gramme, die komplexe Strukturen modellieren, in der gleichen Weise wie klassische n-Gramme für die Vorhersage von

Elementen und für die Messung der Wahrscheinlichkeit gegebener Instanzen verwendet werden können. Im allgemeinen muss diese Frage verneint werden. Die Auftretenswahrscheinlichkeit einer Instanz I beträgt:

$$P(I) = P\begin{pmatrix} S_{i_7} & S_{i_8} & S_{i_9} \\ S_{i_4} & S_{i_5} & S_{i_6} \\ S_{i_1} & S_{i_2} & S_{i_3} \end{pmatrix} = t_{i_4 i_7} b_{i_5 i_7 i_8} b_{i_6 i_8 i_9} t_{i_1 i_4} b_{i_2 i_4 i_5} b_{i_3 i_5 i_6} a_{i_1} r_{i_1 i_2} r_{i_2 i_3}$$

wobei a_{i_1} die a priori Wahrscheinlichkeit des untersten linken Elementes und $b_{ijk} = P(x = S_k \mid R(S_j, x) \wedge T(S_i, x))$ die bedingte Wahrscheinlichkeit eines Elementes ist, wenn der linke und der untere Nachbar bekannt sind. Da die Ereignisse S_{i_α} nicht unabhängig sind, ist b_{ijk} unbekannt. Es muss also eine andere Möglichkeit gefunden werden, um die n-Gramm-Wahrscheinlichkeiten r_{ij} und t_{ij} zu einem Mass für die Modellkonformität einer Instanz zu vereinen.

4.2 Ein Mass für die Modellkonformität

Wenn ein n-Gramm-Modell für eine Strukturklasse bekannt ist, so kann man sich die Frage stellen, ob gewisse Instanzen $I_1, I_2, \ldots, I_k$ zur Klasse gehören, beziehungsweise welche der Instanzen am ehesten zur Klasse gehört. Wir sprechen dabei von der *Konformität* einer Instanz I zu einer Klasse, die durch das Modell M repräsentiert wird.

Ein bereits diskutiertes Mass bei Markovketten ist die a priori Wahrscheinlichkeit einer Instanz. Sie eignet sich für die Bestimmung der am stärksten modellkonformen Kette aus einer Auswahl von Ketten (siehe OCR-Nachkorrektur im Abschnitt 4.1.1). Ein Nachteil der Wahrscheinlichkeit als Konformitätsmass ist deren starke Sensitivität für die Kettenlänge. Da im allgemeinen die Koeffizienten $a_{ij} < 0$ und die Anzahl der Koeffizienten zur Kettenlänge proportional ist, haben lange Ketten tendenziell tiefere Wahrscheinlichkeiten als kurze, auch dann, wenn sie völlig legale Instanzen aus der Klasse darstellen. Die Wahrscheinlichkeit entspricht also nicht dem geforderten, intuitiven Mass für die Konformität.

Ein alternatives Mass zur a priori Wahrscheinlichkeit ist die aus Anwendungen der Fuzzy-Logik bekannte *Generalized-Means*-Funktion h_α für die Aggregation von Fuzzy-Membership Werten [KF92]:

$$h_\alpha : [0,1]^d \to [0,1]$$
$$h_\alpha(a_1, a_2, \ldots, a_d) = \left(\frac{a_1^\alpha + a_2^\alpha + \ldots + a_d^\alpha}{d}\right)^{\frac{1}{\alpha}}$$

Die Funktion erfüllt die vier Axiome für Aggregationsfunktionen:

1. Randbedingungen: $h_\alpha(0, 0, \ldots, 0) = 0$ und $h_\alpha(1, 1, \ldots, 1) = 1$

	fuzzy AND	harmonic mean	geometric mean	arithmetic mean	fuzzy OR
$\alpha =$	$-\infty$	-1	0	$+1$	$+\infty$

Tabelle 4.1: Aus der Generalized-Means-Funktion durch Variation des Parameters α erhaltene Funktionen.

2. Monoton steigend in allen seinen Argumenten
3. Stetigkeit
4. Symmetrie: $h_\alpha(a_i \mid i \in \mathsf{N}_k) = h_\alpha(a_{p(i)} \mid i \in \mathsf{N}_k)$, für jede Permutation p in N_k

Die Generalized-Means-Funktion besitzt die interessante Eigenschaft, dass sich ihr Verhalten durch Variation von α im Bereich zwischen Fuzzy-And (min-Funktion) und Fuzzy-Or (max-Funktion) stufenlos einstellen lässt, wie Tabelle 4.1 illustriert. Wenn $x \leq 0$ gewählt wird, so hat dies zur Folge, dass ein einzelnes $a_i = 0$ zu einem Funktionswert 0 führt ($h_\alpha(\ldots, 0, \ldots) = 0, \alpha \leq 0$). Bezogen auf unsere Interpretation als Konformitätsmass bedeutet dies, dass ein einzelnes, unkonformes n-Gramm die Gesamtkonformität einer Instanz komplett zerstört. In Anwendungen aber, in denen eine Fehlertoleranz verlangt wird und leichte Abweichungen vom Modell erlaubt sein sollen, ist dies unerwünscht. Andererseits ergeben zu grosse positive Werte von α zu eine zu tolerante Messung der Konformität, da die grössten Werte in $(a_1, \ldots, a_k)$ den Wert der Gesamtkonformität dominieren und gegen 1 treiben. Für unsere Anwendungen haben sich Werte für α zwischen 0 und 1 als praktikabel erwiesen. Diesen Sachverhalt verdeutlicht Tabelle 4.2, in der die Konformitätswerte von legalen (1)–(4) und illegalen Instanzen (5) und (6) aufgelistet sind. Die illegalen Instanzen wurden so gewählt, dass sie mit einer Elementvertauschung nur leicht von einer legalen abweichen. Trotzdem werden, der intuitiven Forderung gehorchend, für illegale Instanzen meist niedrigere Konformitätswerte berechnet als für legale. Den Berechnungen wurden die theoretischen Bigramm-Werte zugrundegelegt.

Aus der Tabelle ist auch zu entnehmen, dass das Modell M_{XY} die Klasse von 0-1-Gittern offensichtlich besser repräsentiert als M_{RT} (siehe dazu die Diskussion der Entropie eines Modells auf Seite 58).

4.3 Ein n-Gramm basiertes Dokumentmodell

Im folgenden Abschnitt wird unser konkreter Ansatz für die Modellierung von Dokumentstrukturen mit generalisierten n-Grammen vorgestellt. Zuvor erfolgt

Instanz		Modell M_{RT}			Modell M_{XY}		
		$\alpha =$			$\alpha =$		
		0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1
1)	1 1 1 1 1 1 0 1 1	0.858	0.881	0.900	0.951	0.954	0.958
2)	1 1 1 0 1 1 0 0 1	0.737	0.770	0.800	0.859	0.867	0.875
3)	0 1 1 0 0 1 0 0 0	0.737	0.770	0.800	0.859	0.867	0.792
4)	0 0 1 0 0 0 0 0 0	0.858	0.881	0.900	0.951	0.954	0.708
5)	1 1 1 1 1 1 0 0 0	0.795	0.825	0.850	0	0.496	0.667
6)	1 1 1 1 0 1 0 0 1	0	0.631	0.716	0	0.335	0.542

Tabelle 4.2: Konformitätswerte von Mustern für die Modelle M_{RT} und M_{XY} bei verschiedenen Werten für α.

eine genaue Definition der spezifischen logischen und physischen Struktur die modelliert werden soll.

4.3.1 Die spezifische Struktur

Die spezifische Dokumentstruktur beschreibt die Struktur eines einzelnen Dokuments. Sie besteht aus einem physischen Teil, der das Layout des Dokuments beschreibt, und aus einem logischen Teil für die logische Gliederung des Dokuments (vgl. Kapitel 2 über Dokumentstrukturen). Wir definieren nun präziser die physische und logische Struktur, so wie wir sie für die Modellierung, für das Lernen und für die Dokumenterkennung verwenden werden.

Die physische Struktur

Die physische Struktur eines Dokuments ist ein Baum. Die Knoten darin besitzen Attribute, die vom Typ des Knoten abhängen. Es gibt genau sechs Knotentypen: Volume, Page, Block, Line, Word und Sign. Die Wurzel des Baumes ist ein Volume-Knoten. Er enthält einen oder mehrere Page-Knoten, die den Seiten entsprechen. Diese sind ihrerseits in Block-Knoten aufgeteilt. Wir versuchen, dem Begriff „Block" eine Definition zu geben, die weder zirkulär, noch abhängig von logischen Strukturelementen ist. Ein Textblock ist ein nicht mehr vergrösserbarer, rechteckiger Bereich in einer Seite, dessen Folge von ungeteilten Textzeilen der Lesefolge von oben nach unten entspricht[1].

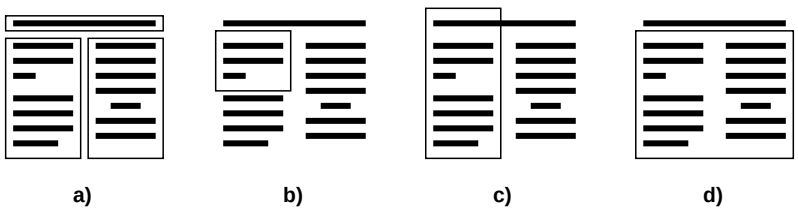


Abbildung 4.4: Korrekte a) und inkorrekte b) bis d) Seitenaufteilung in Blöcke (die horizontalen Balken stellen Textzeilen dar).

Die in Abbildung 4.4 unter a) gezeigte Blockunterteilung entspricht unserer Definition, nicht jedoch b) bis d): b) weil der Block nicht maximal ist, c) weil die oberste Textzeile durch die Blockgrenzen geteilt wird und d) weil die Zeilenfolge von oben nach unten nicht der Lesefolge entspricht.

Die typischen Attribute eines Blocks sind die Position und das Ausmass des umschriebenen Rechtecks. Blöcke sind weiter unterteilt in Textzeilen (Line-Knoten). Deren typische Attribute sind linke und rechte Einrückungen bezüglich des umschriebenen Blocks sowie die Distanz zur vorherigen und nachfolgenden Zeile. Textzeilen enthalten Wörter, die als Attribut deren Zeichensatz besitzen. Wörter bestehen schliesslich aus Zeichen (Sign-Knoten), mit deren Zeichencode als Attribut.

Um Beziehungen zwischen physischen Elementen später mit n-Grammen ausdrücken zu können, müssen die Attribute aller physischen Knoten, die für die Dokumenterkennung relevant sind, diskretisiert und in eine Nominalskala abgebildet werden. Zeilen können beispielsweise klassifiziert werden in „links-

[1] Obwohl der Begriff „Lesefolge" (reading order) ein Terminus der logischen Struktur ist, kann in den meisten Fällen deren Detektierung bei der Erkennung der physischen Struktur vollzogen werden. Wir beschränken uns hier auf diese Fälle.

und rechtsbündig", „links eingerückt" oder „rechts eingerückt". Jeder Knotentyp besitzt seine eigenen Klassen. Die Klassen bilden das *physische Label*, bestehend aus den Elementen $\Phi = \varphi_1, \varphi_2, \ldots, \varphi_{m_\Phi}$, von denen jeder physische Knoten genau eines besitzt.

Die logische Struktur

Die logische Struktur eines Dokuments ist ein Baum. Die Knoten darin besitzen keine eigentlichen Attribute sondern lediglich den Namen des Knotens: sein *logisches Label*. Die logischen Label entsprechen den logischen Entitäten $\Lambda = \lambda_1, \lambda_2, \ldots, \lambda_{m_\Lambda}$ in einem Dokument, also beispielsweise „Titel", „Absatz" oder „Aufzählung".

Die Verbindung zwischen logischer und physischer Struktur

Für die Repräsentation der Zusammenhänge zwischen der logischen und der physischen Struktur eines Dokuments hat sich für unsere Anwendung der Erkennung folgender Ansatz bewährt: Die logischen Entitäten werden als eine von physischen Entitäten konstruierte Sequenz betrachtet. Jeder logische Knoten wird mit dem ersten und dem letzten physischen Knoten verbunden. Dabei gelten die Regeln:

1. Die erste und die letzte physische Entität einer logischen Entität müssen vom gleichen Typ sein.

2. Alle logischen Entitäten mit dem gleichen logischen Label müssen auf physische Entitäten des gleichen Typs verweisen.

3. Alle logischen Entitäten mit dem gleichen logischen Label verweisen auf physische Entitäten, die auf dem höchstmöglichen Niveau im physischen Baum liegen. Die logischen Entitäten müssen gewissermassen auf einen „grössten gemeinsamen Nenner" gebracht werden.

Die folgenden Beispiele verdeutlichen die drei Regeln. Ein logischer Knoten darf nicht auf eine Zeile als erste und ein Wort als letzte physische Entität verweisen (Regel 1). Es darf in der ganzen Dokumentklasse keine logischen Entitäten wie „Absatz" geben, von denen einige auf Zeilen und andere auf Wörter verweisen (Regel 2). Die logische Entität „Absatz" darf nicht auf Zeichen verweisen, da die grössten gemeinsamen physischen Entitäten Textzeilen sind (Regel 3).

Zwei Abbildungen illustrieren anhand desselben Dokumentausschnitts die eingeführten Konzepte. Abbildung 4.5 zeigt logische Labels, die Textfragmenten zugeordnet sind. Abbildung 4.6 stellt die physische und die logische Ebene dar.

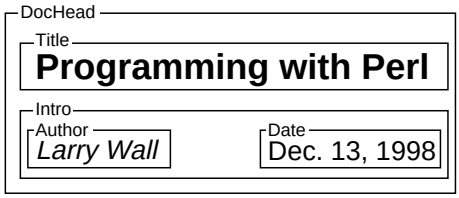


Abbildung 4.5: Die logische Struktur dargestellt als Labels auf einem Dokumentausschnitt.

Darin sind die Verbindungen sichtbar gemacht, die von jedem logischen Element ausgehen.

4.3.2 Die generische Struktur

Die beschriebene Dokumentstruktur mit den beiden verbundenen Bäumen soll nun durch generalisierte n-Gramme modelliert werden. Dies erfolgt in analoger Weise wie im Modellierungsbeispiel für ein zweidimensionales Gitter von binären Elementen.

Das Modell besteht aus zwei Teilen: Der erste Teil beschreibt die Beziehung zwischen den logischen Elementen, der zweite die Beziehung zwischen den logischen und physischen Elementen. Eine Modellierung der Beziehung zwischen physischen Elementen als solche ist nicht notwendig, da wir für die Erkennung davon ausgehen, dass die physische Struktur vollständig gegeben ist, und darauf mit Hilfe des statistischen Modells die logische Struktur aufgebaut wird.

Die Modellierung der logischen Struktur

Für die Modellierung der logischen Struktur definieren wir die Relationen $L(\lambda, \mu, \ldots)$, $R(\lambda, \mu, \ldots)$, $T(\lambda, \mu, \ldots)$, $B(\lambda, \mu, \ldots)$ und $E(\lambda, \mu, \ldots)$. Sie bezeichnen Nachbarschaftsbeziehungen von Knoten in einer Baumstruktur, wie in Tabelle 4.3 definiert. Damit lassen sich die folgenden Trigramme definieren:

$$
\begin{aligned}
r_{\lambda\mu\nu} &= P(x = \nu \mid R(\lambda, \mu, x)) \\
l_{\lambda\mu\nu} &= P(x = \nu \mid L(\lambda, \mu, x)) \\
t_{\lambda\mu\nu} &= P(x = \nu \mid T(\lambda, \mu, x)) \\
b_{\lambda\mu\nu} &= P(x = \nu \mid B(\lambda, \mu, x)) \\
e_{\lambda\mu\nu} &= P(x = \nu \mid E(\lambda, \mu, x))
\end{aligned}
$$

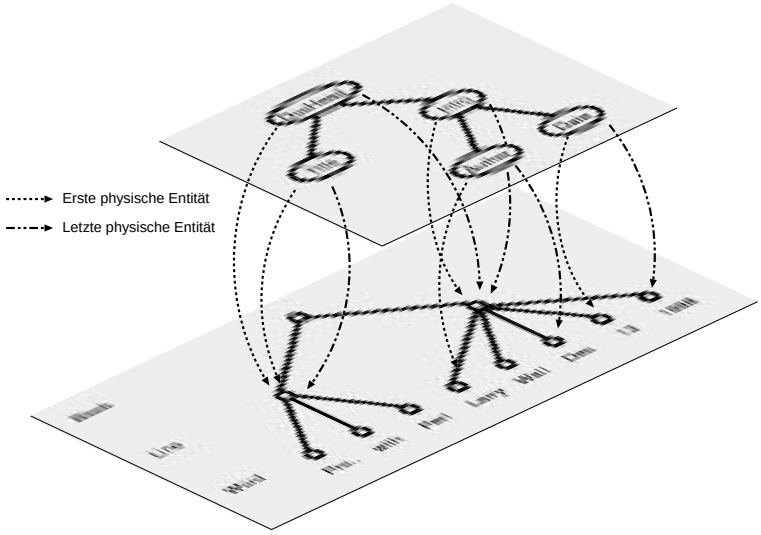


Abbildung 4.6: Die logische und physische Struktur dargestellt als zwei miteinander verbundene Baumstrukturen.

$$\text{mit } \lambda, \mu \in \Lambda,\ \nu \in \Lambda \cup \lambda_0.$$

Wenn wir eines der Trigramme exemplarisch herausgreifen, so bedeutet $r_{\lambda\mu\nu}$ die Wahrscheinlichkeit, einen logischen Knoten ν anzutreffen, wenn diesem ein Knoten μ vorangeht und diesem wiederum λ. In der Definition der Trigramme kann ein logisches Pseudoelement λ_0 vorkommen, welches für ein Grenzelement in der Baumstruktur steht. So bezeichnet $r_{\lambda\mu\lambda_0}$ die Wahrscheinlichkeit, den Knoten μ anzutreffen, wenn diesem ein Knoten λ vorangeht, und er selbst der letzte direkte Nachkomme seines Vorgängers ist.

Obgleich in der Folge nur Tri- und Bigramme besprochen werden, sei hier der Vollständigkeit halber am Beispiel von $r_{\mu_1 \ldots \mu_n}$ die Definition eines allgemeinen n-Gramms gegeben:

$$r_{\mu_1 \ldots \mu_n} = P(x = \mu_n \mid R(\mu_1, \ldots, \mu_{n-1}, x)),$$

bezeichnet die Wahrscheinlichkeit μ_n anzutreffen, wenn ihm die Sequenz μ_1, $\ldots, \mu_{n-1}$ vorangeht ($\mu_1, \ldots, \mu_{n-1} \in \Lambda, \mu_n \in \Lambda \cup \Lambda_0$). Die restlichen n-Gramme sind analog definiert.

Ein n-Gramm-Modell der logischen Struktur besteht aus allen n-, $n-1$-,

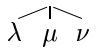
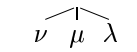
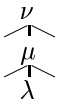
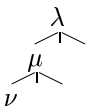
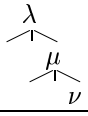
Baummuster	Beziehung zwischen λ, μ und ν	Notation
	Der Knoten ν ist der rechte Nachbar des Musters λ μ.	$R(\lambda, \mu, \nu)$
	Der Knoten ν ist der linke Nachbar des Musters μ λ.	$L(\lambda, \mu, \nu)$
	Der Knoten ν ist der direkte Vorgänger von μ, der der direkte Vorgänger von λ ist.	$A(\lambda, \mu, \nu)$
	Der Knoten ν ist der erste direkte Nachfolger von μ, der der erste direkte Nachfolger von λ ist.	$B(\lambda, \mu, \nu)$
	Der Knoten ν ist der letzte direkte Nachfolger von μ, der der letzte direkte Nachfolger von λ ist.	$E(\lambda, \mu, \nu)$

Tabelle 4.3: Notation für die Beschreibung von lokalen Beziehungen in einem Baum.

..., 2-Grammen; Das Trigramm-Modell ist folglich

$$\begin{aligned} M_\Lambda &= (r_{\lambda\mu\nu}, l_{\lambda\mu\nu}, t_{\lambda\mu\nu}, b_{\lambda\mu\nu}, e_{\lambda\mu\nu}, r_{\mu\nu}, l_{\mu\nu}, t_{\mu\nu}, b_{\mu\nu}, e_{\mu\nu}) \\ &\quad \forall\, i, j \in \Lambda,\ k \in \Lambda \cup \lambda_0. \end{aligned}$$

die Kollektion aller Trigramme und Bigramme für die Beschreibung der Baumstruktur.

Die Modellierung des Zusammenhangs zwischen logischer und physischer Struktur

Für die Modellierung des Zusammenhangs zwischen logischer und physischer Struktur definieren wir die Relationen I^f, I^l, H^f und H^l. Sie bezeichnen die Nachbarschaftsbeziehungen zwischen logischen und physischen Knoten in ihren Baumstrukturen und sind in Tabelle 4.4 definiert. Damit lassen sich die folgenden Trigramme definieren:

$$\begin{aligned} i^f_{\varphi\vartheta\nu} &= P(x = \nu \mid I^f(\varphi, \vartheta, x)) \\ i^l_{\varphi\vartheta\nu} &= P(x = \nu \mid I^l(\varphi, \vartheta, x)) \end{aligned}$$

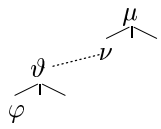
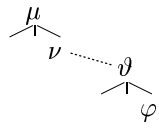
Baummuster	Beziehung zwischen φ, ϑ, ν und μ	Notation
	Der Knoten φ ist der erste direkte Nachfolger von ϑ, und ϑ ist die erste physische Entität von ν.	$I^f(\varphi, \vartheta, \nu)$
	Der Knoten μ hat als ersten direkten Nachfolger ν, und ν hat als erste physische Entität ϑ.	$H^f(\mu, \nu, \vartheta)$
	Der Knoten φ ist der letzte direkte Nachfolger von ϑ, und ϑ ist die letzte physische Entität von ν.	$I^l(\varphi, \vartheta, \nu)$
	Der Knoten μ hat als letzten direkten Nachfolger ν, und ν hat als letzte physische Entität ϑ.	$H^l(\mu, \nu, \vartheta)$

Tabelle 4.4: Notation für die Beschreibung von Beziehungen zwischen zwei Bäumen.

$$
\begin{aligned}
h^f_{\mu\nu\vartheta} &= P(x = \vartheta \mid H^f(\mu, \nu, x)) \\
h^l_{\mu\nu\vartheta} &= P(x = \vartheta \mid H^l(\mu, \nu, x)) \\
&\quad \text{mit } \varphi, \vartheta \in \Phi,\ \nu, \mu \in \Lambda,
\end{aligned}
$$

und analog die Bigramme:

$$
\begin{aligned}
i^f_{\vartheta\nu} &= P(x = \nu \mid I^f(\vartheta, x)) \\
i^l_{\vartheta\nu} &= P(x = \nu \mid I^l(\vartheta, x)) \\
h^f_{\nu\vartheta} &= P(x = \vartheta \mid H^f(\nu, x)) \\
h^l_{\nu\vartheta} &= P(x = \vartheta \mid H^l(\nu, x)) \\
&\quad \text{mit } \vartheta \in \Phi,\ \nu \in \Lambda.
\end{aligned}
$$

Ein n-Gramm-Modell M_Φ des Zusammenhangs zwischen physischer und logischer Struktur besteht aus allen n-, $n-1$-, ... 2-Grammen i^f, i^l für die Beschreibung eines logischen Knotens gegeben den physischen Kontext und h^f, h^l für die Beschreibung eines physischen Knotens gegeben den logischen Kontext. Das Trigramm-Modell ist folglich:

$$
\begin{aligned}
M_\Phi &= (i^f_{\varphi\vartheta\nu}, i^l_{\varphi\vartheta\nu}, h^f_{\mu\nu\vartheta}, h^l_{\mu\nu\vartheta}, i^f_{\vartheta\nu}, i^l_{\vartheta\nu}, h^f_{\nu\vartheta}, h^l_{\nu\vartheta}) \\
&\quad \forall\ \varphi, \vartheta \in \Phi,\ \mu, \nu \in \Lambda,
\end{aligned}
$$

die Kollektion aller Trigramme und Bigramme für die Beschreibung der beiden Baumstrukturen.

4.4 Zusammenfassung

Statistische Methoden sind in den letzten Jahren auf breiter Front zum Einsatz gelangt, typischerweise dort, wo grosse Mengen unsicherer Information verarbeitet werden müssen, und wo ein fehlertolerantes Systemverhalten gefordert wird. n-Gramm-Modelle stellen einen statistischen Ansatz für die Modellierung von Ereignissequenzen dar. Wir haben eine Generalisierung der Interpretation von n-Grammen vorgeschlagen, die es erlaubt, nahezu beliebige Strukturen (z.B. Netzwerke von Ereignissen) zu modellieren.

Mit n-Gramm-Modellen lässt sich zudem die Modellkonformität von Instanzen einer Struktur messen. Ein geeignetes Mass ist die Generalized-Means-Funktion, was in der Evaluation später unter Beweis gestellt wird. Wir haben schliesslich einen Ansatz vorgestellt, wie mit n-Grammen sowohl die logische Struktur einer Dokumentklasse als auch ihr Zusammenhang mit der physischen Struktur modelliert werden können.

Kapitel 5

Lernen

Die Fähigkeit zu lernen wird gemeinhin als eine komplexe Funktion menschlicher und tierischer Gehirne betrachtet und gilt daher als ein wichtiges Kriterium für natürliche Intelligenz. So zeigen sich Informatiklaien nicht selten sehr erstaunt, wenn sie von lernfähigen Computersystemen vernehmen. Dieser Sicht vom Lernen als undurchschaubar komplexer, biologisch neuronaler Prozess, steht die etwas entmystifizierende Definition von Herbert Simon (aus [Lug94]) gegenüber:

> *Learning is any change in a system that allows it to perform better the second time on repetition of the same task drawn from the same population.*

Es ist also sinnvoll, auch dann von Lernen zu sprechen, wenn ein Programm seine Parameter so verändert, dass es eine Aufgabe besser bewältigen kann, selbst wenn es primitiverweise lediglich einige zufällige Parametervarianten gegeneinander vergleicht.

Beim Lernen von Computerprogrammen geht es spezifisch darum, das Systemverhalten anzupassen oder zu verbessern, ohne es umprogrammieren zu müssen. Man unterscheidet zwei Arten von Lernen: supervisiertes und unsupervisiertes Lernen. Beim supervisierten Lernen leitet ein Trainer den Lernvorgang, indem er dem System Beispiele liefert und sein Verhalten validiert. Beim unsupervisierten Lernen entscheidet das System aufgrund eines einprogrammierten Metamodells selbständig, was es zu welchem Zeitpunkt lernt.

Lernfähige Systeme, also solche, die sich ihre Modelle selbständig aus Lerndaten aufbauen, bieten die folgenden Vorteile.

- Sie können grosse Mengen an Lerndaten automatisch oder halbautomatisch verarbeiten. Eine aufwendige manuelle Erstellung eines Modells entfällt.

- Sie können die Konsistenz gelernter Modelle eher sicherstellen als bei manuell erstellten. Die manuelle Erstellung ist fehleranfällig und wird als mühsam empfunden.
- Sie können entwickelt werden, auch wenn das konkrete Problem zum Zeitpunkt der Programmierung noch gar nicht bekannt ist.
- Sie sind ökonomischer, weil sie nur einmal programmiert zu werden brauchen und dennoch auf ein breites Problemspektrum anwendbar bleiben.

Als Nachteil muss die im allgemeinen niedrigere Leistungsfähigkeit gewertet werden. Da Handoptimierungen nicht vorgesehen sind, arbeiten lernfähige Systeme in der Regel langsamer und weniger zuverlässig als hart programmierte. Dies ist der Preis, der für die hohe Flexibilität zu bezahlen ist.

5.1 Lernen von n-Gramm-Modellen

Ein bedeutender Vorteil von n-Gramm-Modellen ist, dass sie sehr leicht durch supervisiertes Lernen aus Beispielen generiert werden können. Wir unterscheiden zwei Phasen des Lernens: Beim *initialen Lernen* wird ein zunächst leeres Modell erzeugt, welches durch Lernen von Beispielen initialisiert wird. Die initiale Lernphase generiert ein möglicherweise unvollständiges Modell, das aber bereits für Erkennungsaufgaben taugt. Das Modell kann nachträglich durch *inkrementelles Lernen* vervollständigt werden, womit sich die Erkennungsleistung entsprechend verbessert. Grundsätzlich können auf eine initiale beliebig viele inkrementelle Lernphasen folgen. Zwischen den Lernphasen lässt sich das Modell weiterhin für die Erkennung verwenden.

5.1.1 Initiales Lernen

Gegeben sei eine Serie von Lernbeispielen K_1, K_2, ..., auch Samples genannt. Es soll in einer initialen Lernphase ein neues n-Gramm-Modell durch eine Analyse der Samples erzeugt werden. Das Lernen eines n-Gramm-Modell besteht lediglich darin, die bedingten Wahrscheinlichkeiten der n-Gramme zu schätzen.

Betrachten wir als Beispiel die Markovketten aus Kapitel 4. Wir bezeichnen die Häufigkeit einer Ereignisfolge $S_iS_jS_k$ im Sample K_h mit $F(S_iS_jS_k, K_h)$. Die Wahrscheinlichkeit der Trigramme a_{ijk} können, basierend auf dem Sample K_h, geschätzt werden mit:

$$a_{ijk} \quad = \quad P(q_t = S_k \mid q_{t-1} = S_j, q_{t-2} = S_i) = \frac{P(S_iS_jS_k)}{P(S_iS_j)}$$

$$\doteq \frac{F(S_iS_jS_k, K_h)}{\sum_{\forall u\in[0,m]} F(S_iS_jS_u, K_h)} = \hat{a}_{ijk}.$$

Für die Schätzung der Trigramme müssen somit nur die Häufigkeiten der Sequenzen der Länge $n = 3$ bekannt sein. Die Häufigkeiten der Sequenzen sind algorithmisch durch einfaches Zählen im Sample K_n zu bestimmen.

Selbstverständlich ist es auch möglich, mehrere Samples gleichzeitig zu lernen. Wir ersetzen dazu lediglich den Term $F(S_iS_jS_k, K_h)$ durch $F(S_iS_jS_k, K_{h_1}, \ldots, K_{h_s})$, wobei gilt:

$$F(S_iS_jS_k, K_{h_1}, \ldots, K_{h_s}) = \sum_{l=1,}^{s} F(S_iS_jS_k, K_{h_l}),$$

die Summe der Häufigkeiten der Sequenz $S_iS_jS_k$ in den Samples $K_{h_1}, \ldots, K_{h_s}$.

Diese Lernmethode für n-Gramme von Markovketten ist direkt auf das Lernen von generalisierten n-Grammen komplexer Strukturen übertragbar. Greifen wir als Beispiel die Trigramme $r_{\lambda\mu\nu}$ heraus, welche die Wahrscheinlichkeiten von logischen Entitäten beschreiben. Ist deren linker Kontext desselben Unterbaums bekannt, so ist die Schätzung $\hat{r}_{\lambda\mu\nu}$ gegeben durch:

$$r_{\lambda\mu\nu} \doteq \hat{r}_{\lambda\mu\nu} = \frac{F(R(\lambda\mu\nu), K_{h_1}, \ldots, K_{h_s})}{\sum_{\forall\kappa\in\Lambda\cup\lambda_0} F(R(\lambda\mu\kappa), K_{h_1}, \ldots, K_{h_s})},$$

wobei $F(R(\lambda\mu\nu), K_{h_1}, \ldots, K_{h_s})$ die kumulierte Häufigkeit des Musters $R(\lambda\mu\nu)$ in den Samples $K_{h_1}, \ldots, K_{h_s}$ bezeichnet.

Die weiteren Trigramme $l_{\lambda\mu\nu}$, $t_{\lambda\mu\nu}$, $b_{\lambda\mu\nu}$, $e_{\lambda\mu\nu}$ für die logischen und $i^f_{\varphi\vartheta\nu}$, $i^l_{\varphi\vartheta\nu}$, $h^f_{\mu\nu\vartheta}$, $h^l_{\mu\nu\vartheta}$ für die logisch-physischen Knotenbeziehungen werden analog der obigen Formel geschätzt.

5.1.2 Inkrementelles Lernen

Das Ziel des inkrementellen Lernens ist, ein bestehendes Modell nachträglich mit zusätzlichen Lernsamples anzureichern und somit sukzessive zu vervollständigen. Dies ist in einem n-Gramm-Modell leicht zu realisieren, wenn aus der letzten Lerngeneration g nicht nur die n-Gramm-Wahrscheinlichkeiten sondern auch die Musterhäufigkeiten $F(\ldots)$ zurückbehalten werden. Ein neues Modell $M^{(g+1)}$ der Generation $g + 1$, das mit einem Sample K_{s+1} angereichert wird, lässt sich ausdrücken durch (wiederum vereinfacht dargestellt anhand der Trigramme $r_{\lambda\mu\nu}$):

$$\hat{r}^{(g+1)}_{\lambda\mu\nu} = \frac{\hat{r}^{(g)}_{\lambda\mu\nu} \sum_\kappa F(R(\lambda\mu\kappa), K_{h_1}, \ldots, K_{h_s}) + F(R(\lambda\mu\nu), K_{h_{s+1}})}{\sum_\kappa F(R(\lambda\mu\kappa), K_{h_1}, \ldots, K_{h_{s+1}})}.$$

Der obige Ausdruck sieht komplizierter aus, als er tatsächlich ist. Wenn man sich vor Augen führt, dass ihm die Beziehung für die inkrementelle Berechnung von Mittelwerten $\overline{m}^{(n+1)} = \frac{\overline{m}^{(n)} n + a_{n+1}}{n+1}$ zugrundeliegt, wird seine Struktur transparenter.

5.2 Lernen von Dokumentausschnitten

Wenn wir bisher vom Lernen von Samples gesprochen haben, so setzten wir stillschweigend voraus, dass jedes Sample K_h einer vollständigen Dokumentinstanz entspricht. Das muss jedoch nicht notwendigerweise der Fall sein. Wir erinnern uns an die Anforderung an das System, dass ein Benutzer lediglich diejenigen Dokumentausschnitte manuell korrigieren muss, an denen es zuvor gescheitert ist. Es sollte demnach möglich sein, auch Fragmente von Dokumenten zu lernen, die spezifische, neue Information enthalten und in das Modell integriert werden.

Nun stellt es grundsätzlich keine Schwierigkeit dar, lassen wir als Samples K_h auch Fragmente von Dokumenten zu, wie beispielsweise einen einzelnen Absatz oder eine Aufzählungsliste. Es können weiterhin dieselben Algorithmen und Berechnungsformeln verwendet werden, wie für das inkrementelle Lernen. Dabei muss aber beachtet werden, dass sich bei intensivem Lernen von Fragmenten die geschätzten n-Gramm-Wahrscheinlichkeiten allmählich verzerren. Das liegt daran, dass nur Muster gelernt werden können, die sich innerhalb der Fragmente befinden, nicht aber die Beziehung ihrer Randknoten mit Knoten ausserhalb des Fragments. Im allgemeinen stellt dies aber ein untergeordnetes Problem dar, denn der Erkennungsalgorithmus ist nicht auf präzise Werte der n-Gramm-Wahrscheinlichkeiten angewiesen.

5.3 Die Entropie von Modellen

Wir haben gezeigt, wie Dokumentstrukturen mit n-Grammen modelliert und gelernt werden können. Bevor wir uns nun anschicken, das n-Gramm-Modell praktisch anzuwenden, versuchen wir auf eine theoretische Weise zu bestimmen, wie gut das Modell die Dokumentstruktur überhaupt zu beschreiben vermag.

Dazu dient uns das Konzept der Entropie [Cha93, KF92], das von Shannon im Zusammenhang mit der Informationstheorie eingeführt wurde. Die Entropie misst die Verteilung der Wahrscheinlichkeiten von Zufallsvariablen. Die Entropie H ist eine Funktion der Form:

$$H : \mathcal{P} \rightarrow [0, \infty[,$$

wobei $\mathcal{P}$ die Menge aller Wahrscheinlichkeitsverteilungen auf endlichen Mengen

bezeichnet. H ist demnach definiert als

$$H(p(x) \mid x \in X) = -\sum_{x \in X} p(x) \log p(x),$$

wobei $(p(x) \mid x \in X)$ eine Wahrscheinlichkeitsverteilung auf der endlichen Menge X bezeichnet.

Die Entropie hat die Eigenschaft, niedrige Werte anzunehmen, wenn die Wahrscheinlichkeiten $p(x)$ nahe bei 0 oder 1 liegen. Es ist intuitiv einsehbar, dass in einem Modell diejenigen n-Gramme besonders aussagekräftig sind, deren Werte nahe bei 0 oder 1 liegen, weil sie sehr dezidierte Aussagen über benachbarte Entitäten für einen gegebenen Kontext machen. Ein n-Gramm gleich 0 bedeutet, dass die spezifizierte Entität unmöglich in den Kontext passt, während ein n-Gramm gleich 1 bedeutet, dass die spezifizierte Entität mit Sicherheit zum Kontext gehört.

Die Entropie für eine Gruppe von n-Grammen, deren Summe 1 ergibt, ist folglich (am Beispiel des Trigrammes $r_{\lambda\mu\nu}$):

$$H(p_{\lambda\mu}(\kappa) \mid \kappa \in \Lambda \cup \lambda_0) = -\sum_{\kappa \in \Lambda \cup \lambda_0} p_{\lambda\mu}(\kappa) \log p_{\lambda\mu}(\kappa),$$

die Einzelentropie für die rechte Nachbarschaft des Kontexts λ-μ, wobei $r_{\lambda\mu\kappa}$ als diskrete Wahrscheinlichkeitsverteilung

$$p_{\lambda\mu}(\kappa) := r_{\lambda\mu\kappa}$$

aufgefasst wird. Als Mass für die Gesamtentropie eines Modells berechnen wir den gewichteten Mittelwert aller Einzelentropien der n-Gramme.

Wenn wir die Entropie für die Modelle M_{RT} und M_{XY} betrachten, die in Kapitel 4 für die Modellierung der 3×3 Gitterstrukturen definiert wurden, so zeigt sich, dass M_{RT} mit 0.336 eine deutlich höhere Gesamtentropie aufweist als M_{XY} mit 0.159. Dies stützt die Beobachtung in Tabelle 4.2, wonach das Modell M_{RT} mit einer horizontalen und vertikalen Nachbarschaftsrepräsentation die gegebenen Gitterstrukturen schlechter modelliert, als M_{XY} mit einer diagonalen Nachbarschaftsrepräsentation.

5.4 Diskussion

Wir werden in den folgenden Abschnitten einige Fragen diskutieren, die sich im Zusammenhang mit dem Lernen von n-Gramm-Modellen für Dokumentstrukturen aufdrängen.

5.4.1 Grösse des Lernkorpus

Beim Lernen von statistischen Modellen ist generell zu beachten, dass die Datenbasis von Lernsamples, das heisst der Korpus, genügend gross ist, um relevante statistische Daten erheben zu können. Die Grösse eines Lernkorpus kann beim Lernen von n-Grammen erhebliche Ausmasse annehmen, da eine obere Grenze für die Anzahl von n-Grammen m^n beziehungsweise $(m_\Lambda + m_\Phi)^n$ beträgt (Zur Erinnerung: m, m_Λ, m_Φ bezeichnen die Anzahl der Entitäten, bzw. die Grösse des Alphabets).

Daher werden für das Lernen von n-Gramm-Modellen natürlichsprachlicher Texte Korpora in der Grössenordnung von 1 Million Buchstaben herangezogen. Glücklicherweise hat die Erfahrung bei n-Gramm-Modellen für Dokumentstrukturen gezeigt, dass erstaunlich wenige Lerndokumente für akzeptable Erkennungsleistungen nötig sind (siehe Kapitel 7, Evaluation). Dieses Phänomen erklärt sich damit, dass beispielsweise die Bigramm-Modellentropie von Dokumentstrukturen mit 0.47 um fast eine Grössenordnung kleiner ist gegenüber der Bigramm-Entropie von 1.73 von Buchstabenketten natürlicher Sprache[1]. Der überwiegend grösste Teil der n-Gramme eines Dokumentmodells hat einen Wert gleich 0, woraus sich der Schluss ziehen lässt, dass die wenigen n-Gramme ungleich 0 relativ schnell auch in einem kleinen Korpus gelernt werden können.

5.4.2 Lernparameter

Das n-Gramm-Modell für Dokumentstrukturen hat die bemerkenswerte Eigenschaft, praktisch keine Lernparameter zu besitzen. Der einzige Parameter ist n, die Länge der n-Gramme. In den meisten unserer Experimente wurde n konstant auf den Wert 3 gesetzt, was zu durchwegs befriedigenden Resultaten führte. Die praktische Parameterfreiheit des Lernalgorithmus ist als ein erheblicher Vorteil des Ansatzes zu werten.

5.4.3 Lernen von Negativbeispielen

Wir haben bisher nur den Fall betrachtet, in dem das System Samples integriert, die der zu lernenden Dokumentklasse angehören. Manche Systeme bieten zusätzlich die Möglichkeit, Negativbeispiele zu lernen. Sie lernen somit etwas über eine Klasse durch die Analyse von denjenigen Instanzen, die explizit *nicht* zu ihr gehören.

[1] Die Bigramm-Entropie wurde anhand von Texten aus der Prosaliteratur ermittelt, die im Public Domain verfügbar sind. Konkret wurden die Entropiewerte aus einer Konkatenation des reinen ASCII-Textes von Romanen von Arthur Conan Doyle, Jack London und Anthony Trollope berechnet die insgesamt einen Umfang von zirka 14 Millionen Zeichen annahm.

Leider ist es nicht möglich, diese Lernmethode in mathematisch unverfänglicher Weise in n-Gramm-Modellen einzusetzen. Da die Berechnung der n-Gramm-Wahrscheinlichkeiten auf den Häufigkeiten von lokalen Mustern beruht, müssten für die explizite Beschreibung unerlaubter Muster so etwas wie negative Häufigkeiten zugelassen werden. Weil wir aber auf das theoretische Fundament des Lernalgorithmus nicht verzichten wollen, sehen wir vom Lernen von Negativbeispielen ab.

5.4.4 Generalisierung und Spezialisierung

In der zu Beginn des Kapitels zitierten Definition von Lernen spricht Simon vom Lernen als einem Prozess, der zu einer besseren Performance des Systems führt. Wir betrachten etwas genauer, was mit dem Begriff Performance gemeint ist.

- Das System kann ein Sample einer Klasse lernen und aufgrund einer Strukturanalyse entscheiden, dass es ähnliche Instanzen der Klasse zukünftig akzeptieren wird. Das Modell wird permissiver, und wir sprechen von einer *Generalisierung* des Modells.
- Das System kann ein Sample einer Klasse lernen und aufgrund einer Strukturanalyse entscheiden, dass es ähnliche Instanzen der Klasse zukünftig nicht mehr akzeptieren wird. Das Modell wird restriktiver, und wir sprechen von einer *Spezialisierung* des Modells.

Die Generalisierung beim Lernen ist notwendig, weil sich im allgemeinen nicht alle Instanzen einer Klasse erschöpfend aufzählen lassen. Das System soll die Struktur von Instanzen selbständig abstrahieren und somit ähnliche Instanzen zulassen, die nie zuvor gelernt wurden. Die Spezialisierung wirkt dem entgegen, indem sie das Modell so verändert, dass es Instanzen zurückweist, die es zuvor akzeptiert hätte. Die Spezialisierung kann somit die Trennschärfe des Modells erhöhen.

In welcher Weise generalisiert und spezialisiert nun der vorgeschlagene Lernalgorithmus für n-Gramme? Zunächst ist festzuhalten, dass die Begriffe Generalisierung und Spezialisierung immer auch im Zusammenhang mit der Interpretation des Modells in der Erkennung zu sehen sind. Wie wir im folgenden Kapitel sehen werden, verwenden wir die n-Gramme in zwei verschiedenen Phasen: einerseits für die schrittweise Konstruktion der logischen Struktur und andererseits für die Messung der Modellkonformität von erzeugten Instanzen der logischen Struktur.

Die erste Phase der Konstruktion der logischen Struktur ist in starkem Masse geleitet von n-Grammen, deren Wert bei 0 oder sehr nahe daran liegt. Wenn wir eine (inkrementelle) Lernphase betrachten, so sehen wir, dass die Anzahl der

n-Gramme, deren Wert 0 ist, niemals zunehmen kann, also monoton fallend ist. Aus der Sicht der Konstruktion der logischen Struktur kann der Lernalgorithmus also nur generalisieren.

Die zweite Phase der Messung der Konformität von Instanzen ist hingegen gleichmässig von allen n-Grammen des Modells beeinflusst, weil die Generalized-Means-Funktion keine Werte speziell behandelt. Da in einer typischen Lernphase gewisse n-Gramme höhere und andere niedrigere Werte annehmen, vollzieht sich beim Lernen gleichzeitig eine Generalisierung und eine Spezialisierung.

Für eine detailliertere Beschreibung des Erkennungsalgorithmus sei auf Kapitel 6 über den Algorithmus für die Dokumenterkennung hingewiesen.

5.5 Zusammenfassung

Lernfähige Systeme bieten gegenüber hart programmierten eine Reihe von Vorteilen: Sie sind langfristig ökonomischer, weil sie auf ein breiteres Problemspektrum anwendbar sind, und sie können mehr Lerndaten verarbeiten als manuell machbar ist. Sie können programmiert werden, noch bevor das Problem in allen Details bekannt ist, und sie können sich bei ihrer Anwendung laufend verbessern und an das Problem anpassen. Die Nachteile sind dagegen die aufwendigere Programmierung und die im allgemeinen niedrigere Systemleistung.

Das vorgeschlagene statistische Modell für Dokumentstrukturen, basierend auf generalisierten n-Grammen, eignet sich sehr gut für ein benutzerassistiertes Lernen. Der Lernvorgang lässt sich unterteilen in eine initiale Lernphase, in der ein erstes Modell erstellt wird, gefolgt von prinzipiell beliebig vielen inkrementellen Lernphasen, in denen das Modell schrittweise verfeinert wird. Zwischen den Lernphasen steht das Modell ohne weiteres für die Erkennung von Dokumentstrukturen zur Verfügung. Es wurde der Begriff der Modellentropie eingeführt, mit der sich die Aussagekraft des Modells und einzelner seiner n-Gramme messen lässt.

Das nun folgende Kapitel zeigt, wie ein einmal gelerntes statistisches Modell für die Erkennung von Dokumentstrukturen angewendet werden kann.

Kapitel 6

Erkennung

Wir haben in den vorangegangenen zwei Kapiteln eine Methode vorgestellt, die es erstens erlaubt, die Struktur von Dokumenten zu modellieren, und zweitens die Möglichkeit bietet, ein Dokumentmodell aus vorgelegten Beispielen zu lernen. Dabei handelt es sich um einen statistischen Ansatz, basierend auf der Repräsentation lokaler Muster mit generalisierten n-Grammen. Bisher ist in der uns bekannten Literatur kein vergleichbares Verfahren dokumentiert worden. Wir werden nun einen Ansatz für die Erkennung der logischen Struktur von Dokumenten unter Verwendung des statistischen Modells vorschlagen.

Unter einer Vielzahl von möglichen Methoden fiel unsere Wahl auf einen heuristischen Ansatz. Heuristiken sind bekannt als Methoden für die Lösung von komplexen Problemen mit Hilfe von Faustregeln, begründeten Vermutungen, intuitiven Bewertungen oder gesundem Menschenverstand. Da eine Heuristik keinen Erfolg garantiert, kann die Suche nach einer Lösung auch fehlschlagen oder ein vom Optimum entferntes Resultat liefern. Heuristiken werden häufig da herangezogen, wo zuverlässige Algorithmen zu zeitaufwendig oder unbekannt sind, oder nicht existieren [Pea85a, Ree95]. Für uns waren die folgenden Beweggründe ausschlaggebend, die Wahl eines heuristischen Ansatzes zu treffen.

- Die im Modell enthaltene Information ist unscharf und daher ungeeignet für exakte, formale Methoden.
- Die Struktur von realen Dokumenten weist häufig Abweichungen von einem idealen Modell auf. Weder exakte Modelle noch exakte Erkennungsmethoden werden realen Dokumenten gerecht.
- Trotz, oder gerade wegen der Unschärfe realer Dokumente weisen ihre Strukturen in hohem Masse Redundanzen auf. Wenn ein Titel beispielsweise infolge eines Fehlers nicht fett gedruckt ist, so bleibt er aufgrund

einer Reihe anderer typographischer Attribute als Titel erkennbar. Heuristische Systeme können besser mit solchen widersprüchlichen Daten umgehen, als exakte.

In diesem Kapitel wird zuerst der heuristische Erkennungsalgorithmus vorgestellt. Darauf folgen eine Beschreibung von notwendigen Optimierungen und schliesslich eine Diskussion einiger herausgegriffener Eigenschaften des Ansatzes.

6.1 Ein heuristischer Erkennungsalgorithmus

6.1.1 Grundprinzip

Wir werden zunächst das grobe Funktionsprinzip des Algorithmus erläutern und anschliessend präziser auf die einzelnen Elemente eingehen.

Bevor die Erkennung beginnen kann, müssen zwei Dinge gegeben sein. Wir gehen erstens davon aus, dass für jede Seite des zu erkennenden Dokuments ein schwarz-weiss-Bild vorliegt, und zweitens, dass ein Modell für die Dokumentklasse zur Verfügung steht, aus der das Dokument stammt.

Aus den Dokumentbildern wird die physische Struktur generiert. Da wir hierfür ausschliesslich Standardalgorithmen verwenden, und da die Erkennung der physischen Struktur nicht Gegenstand der vorliegenden Arbeit ist, wird sie nur am Rande besprochen.

Die gesuchte Komponente ist die logische Struktur, die zur physischen passt und zum Modell konform ist. Im weiteren sollen auch die Verbindungen zwischen logischer und physischer Struktur gefunden werden. Die Erkennung kann man hier also als Konstruktionsvorgang auffassen. Aus Abbildung 6.1 ist weiter ersichtlich, dass aus dem Erkennungsvorgang auch ein neuer Modellteil resultiert. Damit ist der optional auftretende Umstand gemeint, dass während der Erkennung Benutzerinteraktionen stattfinden, die gelernt und in das Modell integriert werden.

Die logische Struktur wird von ihren Blättern her, also bottom-up konstruiert. Schrittweise werden zuerst Blätter und dann logische Knoten zusammengefasst. Jeder neu konstruierte logische Knoten wird mit der physischen Struktur verbunden, so dass er eine Referenz auf seine erste und seine letzte physische Entität erhält. Die Gruppierung von logischen Knoten wird solange fortgesetzt, bis die Wurzel erreicht und konstruiert ist (Abbildung 6.2).

Bei der Konstruktion von logischen Knoten ist es wahrscheinlich, dass Unsicherheiten oder Mehrdeutigkeiten auftreten. Um dennoch eine global optimale Lösung zu finden, ist es notwendig, verschiedene alternative logische Strukturen parallel zu entwickeln, beziehungsweise Backtracking-Mechanismen zu implementieren, die es erlauben, auf eine einmal gefundene Teillösung zurückzu-

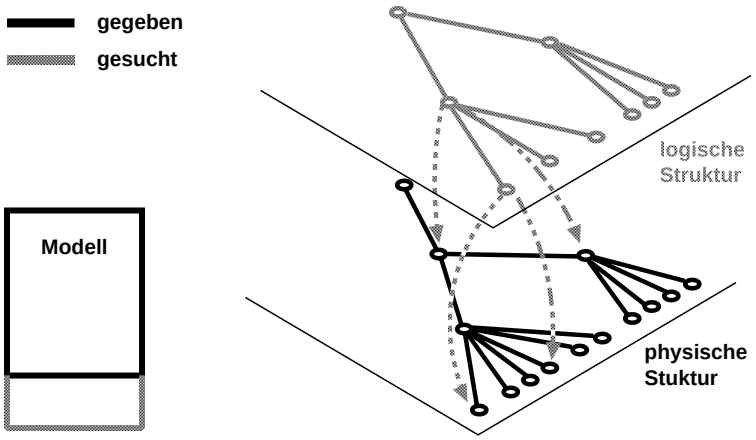


Abbildung 6.1: Vor der Erkennung sind ein Modell und die physische Struktur gegeben. Es resultieren die logische Struktur und ihre Verbindungen zur physischen Struktur sowie optional ein durch inkrementelles Lernen erweitertes Modell.

greifen und andere Konstruktionsalternativen zu explorieren. Dabei müssen die konstruierten Teillösungen laufend unter Anwendung der Generalized-Means-Funktion validiert werden.

Wir betrachten nun kurz die Konstruktion der physischen Struktur und wenden uns detaillierter dem Konstruktionsvorgang der logischen Struktur sowie der Backtracking-Suche nach einer optimalen Lösung zu.

6.1.2 Konstruktion der physischen Struktur

Für die Konstruktion der physischen Struktur werden ausschliesslich Standardalgorithmen verwendet. Der Ausgangspunkt ist das Dokumentbild. Wenn die Dokumente als PostScript- oder PDF-Dateien vorliegen, so können durch Rendern ideale Dokumentbilder erzeugt werden. Bei Vorlagen auf Papier oder Film werden die Bilder mit einem Scanner eingelesen. Eingescannte Bilder werden zusätzlich daraufhin überprüft, ob die Vorlage schief auflag [Bai87, HFD90]. Wenn nötig wird das Bild so rotiert, dass die Zeilen exakt horizontal ausgerichtet sind [Pae89]. Die Segmentierung in eine Hierarchie physischer Elemente erfolgt in einer top-down Vorgehensweise, durch Berechnung und Analyse von

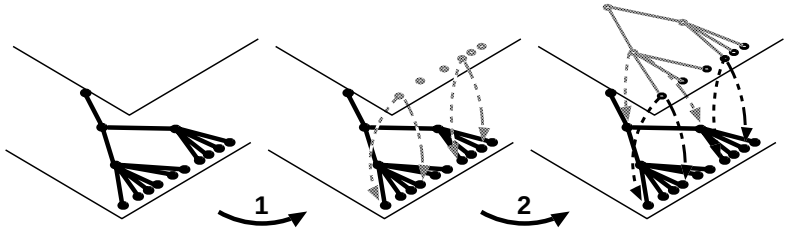


Abbildung 6.2: Die beiden Konstruktionsphasen der logischen Struktur: Die Gruppierung von physischen Entitäten 1) und die Gruppierung von logischen Entitäten 2).

horizontalen und vertikalen Projektionsprofilen sowie der Anwendung des Run-Length-Smoothing Algorithmus [WCW82, AM87].

Im ersten vertikalen Durchgang werden Textspalten detektiert, im zweiten, horizontalen, die Textzeilen und im dritten, vertikalen, in jeder Zeile die Wörter. Die Wörter werden schliesslich in Zeichen segmentiert durch Detektierung ihrer 4- oder 8-konnexen Komponenten. Von jedem Wort wird der Font ermittelt. Je nach Ursprung des Dokumentbildes wird eine von zwei Methoden angewandt. Wurde das Bild durch Rendern einer PostScript- oder PDF-Datei erzeugt, so kann die Fontinformation für jedes Zeichen fehlerfrei aus der Quelldatei extrahiert werden. In allen anderen Fällen wird eine a priori Fonterkennung durchgeführt, die aufgrund optischer Charakteristika den Font ermittelt [Zra95, BI98b]. Eine optische Zeichenerkennung OCR wird nicht vollzogen, da die Textinformation für die Strukturerkennung nicht benötigt wird.

Das Ergebnis der geschilderten Vorgänge ist eine Baumstruktur mit den Knotentypen Volume, Page, Block (=Column), Line, Word und Sign. Damit die Struktur der physischen Knoten im statistischen Modell ausgedrückt werden kann, müssen die Knoten nach ihren physischen Attributen klassifiziert werden. Jedem physischen Knoten wird dabei eines der physischen Labels $\varphi_1, \ldots, \varphi_{m_\Phi}$ zugeordnet. Die Art der Klassifikation variiert mit der zu erkennenden Dokumentklasse. Unsere Wahl der Klassifikation wird daher im Kapitel 7 über die empirischen Tests vorgestellt.

6.1.3 Gruppierung von Knoten

Die Konstruktion der logischen Struktur erfolgt als bottom-up Verfahren von den Blättern zur Wurzel hin. Beginnend bei der physischen Struktur, werden

schrittweise Knoten zur Struktur hinzugefügt. Die Zwischenstufen, der im Aufbau befindlichen Struktur nennen wir *partielle Lösungen*. In jedem Konstruktionsschritt wird geprüft, ob ein neuer logischer Knoten erzeugt werden kann, der eine Gruppe von physischen oder logischen Knoten zusammenfasst. In einem Schritt werden in einer partiellen Lösung 0 oder 1 neuer logischer Knoten erzeugt.

Das Verfahren umfasst zwei Phasen: In der ersten Phase werden physische Knoten zu logischen zusammengefasst, die ihrerseits die Blätter der logischen Baumstruktur bilden. Wenn alle logischen Blätter erzeugt sind, werden diese in einer zweiten Phase schrittweise zusammengefasst, bis die Wurzel erreicht ist und alle logischen Knoten Teil desselben Baums sind.

Anschliessend werden die beiden Phasen der Knotengruppierung detailliert beschrieben. Jeder Abschnitt enthält eine formale und eine textuelle Beschreibung des Algorithmus sowie ein typisches Beispiel. Es sei den individuellen Präferenzen des Lesers überlassen, in welcher Reihenfolge er sich die drei Teile zu Gemüte führen möchte.

Gruppierung physischer Knoten

Im einfachen Fall, wenn sie die physische Struktur vollständig bis auf Zeichen-Entitäten aufgliedert, ist die Konstruktion der logischen Blätterknoten trivial. Da eine 1:1 Beziehung auf der Blätterebene der logischen und physischen Struktur besteht, muss lediglich für jedes physische ein entsprechendes logisches Blatt erzeugt werden.

Es kann hingegen eine deutlich effizientere Erkennung vollzogen werden, wenn die physische Struktur nur bis hin zu den Wort- oder Zeilenentitäten berücksichtigt wird. In diesem Fall wird der folgende Algorithmus verwendet, der zuerst formal und anschliessend verbal dargestellt ist. Die Ziffern, die in beiden Darstellungen den Algorithmusblöcken vorangestellt sind, entsprechen einander. Der Algorithmus beschreibt einen einzelnen Gruppierungsschritt, in dem eine Gruppe von physischen Knoten zu einer logischen Entität zusammengefasst wird.

Definitionen:

- $anc^f(\varphi, \vartheta)$, $anc^l(\varphi, \vartheta)$: die Menge aller möglichen logischen Knoten, die φ als erste (beziehungsweise letzte) physische Entität haben können.

$$
\begin{aligned}
&anc^f : \Phi \times \Phi \to \mathcal{P}(\Lambda) \\
&anc^f(\varphi, \vartheta) = \{\lambda \mid i^f_{\varphi\vartheta\lambda} > 0 \ \vee \ i^f_{\vartheta\lambda} > 0\} \\
&anc^l : \Phi \times \Phi \to \mathcal{P}(\Lambda) \\
&anc^l(\varphi, \vartheta) = \{\lambda \mid i^l_{\varphi\vartheta\lambda} > 0 \ \vee \ i^l_{\vartheta\lambda} > 0\}.
\end{aligned}
$$

```
TYPE  pnode:
              label: Φ
              ancestor: pnode
      sequence:
              content: pnode
              next: sequence

ARG   phys-nodes: sequence

VAR   p: sequence
      first, last: pnode
      κ: Λ

BEGIN
1.    p ← phys-nodes
2.    while („p.content is part of a group“)
              p ← p.next
      first ← p.content

3.    while ¬(anc^f(first.label, first.ancestor.label)
                  ∩anc^l(p.content.label, p.content.ancestor.label) ≠ ∅
                ∧anc^f(p.next.label, p.next.ancestor.label) ≠ ∅)
              p ← p.next
      last ← p.content

4.    foreach κ ∈ (anc^f(first.label, first.ancestor.label)
                  ∩anc^l(last.label, last.ancestor.label)
              „group first... last to logical entity with label κ“
END
```

Abbildung 6.3: Algorithmus für die Gruppierung physischer Knoten.

Dies gilt für alle λ, deren Bigramme $i^f_{\varphi\lambda}$ und Trigramme $i^f_{\varphi\vartheta\lambda}$ (bzw. $i^l_{\ldots}$) einen Wert grösser als 0 besitzen. ϑ ist das physische Label des Vorgängerknotens von φ in der physischen Struktur.

Algorithmus: (siehe auch Abbildung 6.3)

1. Alle physischen Blätter einer partiellen Lösung bilden in Lesefolge eine Sequenz, dargestellt durch das Argument phys-nodes. Mit der Referenz

p wird die Sequenz beginnend mit dem ersten Element von phys-nodes durchlaufen.

2. Suche nach der ersten physischen Entität (first), die noch nicht in eine logische Entität gruppiert wurde.

3. Suche nach einer physischen Entität (last), die zusammen mit first eine Gruppe bilden kann. Solange der aktuelle Knoten p.content nicht eine passende letzte physische Entität sein kann, rücke einen Knoten weiter. Das Abbruchkriterium ist:

   (a) die Menge $anc^f(\text{first}\ldots)$ der logischen Knoten, von denen first die erste Entität sein kann geschnitten mit der Menge $anc^l(\text{p.content}\ldots)$ der logischen Knoten, von denen p.content die letzte Entität sein kann nicht leer ist, und

   (b) dass die Menge $anc^f(\text{p.next}\ldots)$ der logischen Knoten, von denen p.next die erste Entität sein kann nicht leer ist.

4. Fasse die physischen Knoten von first bis last in einen Unterbaum mit der logischen Entität κ als Wurzel zusammen. Die Alternativen für κ entsprechen den Elementen der Schnittmenge der möglichen logischen Vorgänger für first und last.

Bei Betrachtung von Schritt 4. fällt auf, dass bei einer Gruppierung von physischen Knoten mehrere Alternativen für κ erlaubt sind. Im Algorithmus werden alle diese Alternativen realisiert und als konkurrierende Hypothesen, beziehungsweise partielle Lösungen, betrachtet. Die Art, wie mit diesen Hypothesen weiter verfahren wird, ist Gegenstand des Abschnitts 6.1.4 über die Suche nach der optimalen Lösung.

Betrachten wir als Beispiel einer Gruppierung von physischen Knoten die Abbildung 6.4. Die Folge p enthält die Blätter der physischen Struktur, mit p.first= b12. Die Menge der möglichen logischen Knoten, für die b12 als ersten Nachfolger von || die erste physische Entität sein kann ist $\{\mathsf{TT|T}, \mathsf{ST|T}\}$, weil $i^f_{\mathsf{b12,||,TT|T}} > 0$ und $i^f_{\mathsf{b12,||,ST|T}} > 0$. In der Folge p wird solange weitergerückt, bis p.content einen passenden letzten Knoten repräsentiert. Der letzte der Folge von b12 Knoten ist ein solcher Kandidat, weil er mit dem ersten Knoten gemeinsame logische Entitäten besitzt $(anc^f(\mathsf{b12}\ldots) \cap anc^l(\mathsf{b12}\ldots) \neq \emptyset)$ und weil der Folgeknoten n10 eine erste physische Entität sein kann $(\mathsf{anc}^f_{\mathsf{n10,|n,P}} \neq \emptyset)$[1]. Die vier Knoten b12, b12, b12, b12 werden schliesslich zusammengefasst. Da

[1] Der zweite Knoten b12 kann kein Kandidat für den letzten Knoten sein. Da die Menge *anc* ausschliesslich aus Trigrammen konstruiert wurde ($i^f_{\mathsf{b12,||,TT|T}}$ und $i^f_{\mathsf{b12,||,ST|T}}$), müssen für das Auffinden eines letzten physischen Knotens ebenfalls Trigramme verwendet werden ($i^l_{\varphi\vartheta\lambda}$).

Abkürzungen:	Line: top-distance long, bottom-distance long	ll
	Line: top-distance long, bottom-distance normal	ln
	Word: bold font, 12 pt	b12
	Word: normal font, 10 pt	n10
	Table-of-contents Title	TTIT
	Section Title	STIT
Modell:	$i^f_{\mathsf{b12,ll,TTIT}} = 0.05$ $i^l_{\mathsf{b12,ll,TTIT}} = 0.05$ $i^f_{\mathsf{n10,ln,P}} = 0.8$	
	$i^f_{\mathsf{b12,ll,STIT}} = 0.95$ $i^l_{\mathsf{b12,ll,STIT}} = 0.95$	

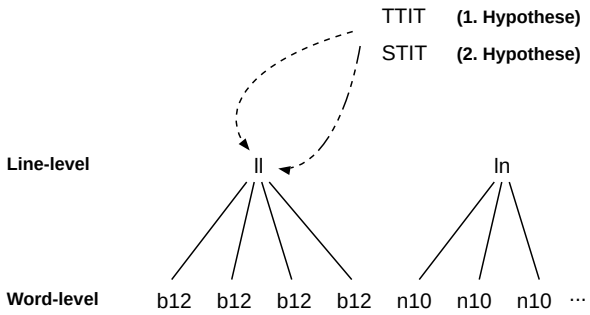


Abbildung 6.4: Beispiel einer Gruppierung von physischen Knoten.

die Schnittmenge der gemeinsamen logischen Vorgängerknoten {TTIT, STIT} ist, ergeben sich zwei neue partielle Lösungen. Die erste gruppiert b12, ..., b12 und erzeugt den logischen Knoten TTIT, und die zweite gruppiert ebenfalls b12, ..., b12 und erzeugt den logischen Knoten STIT.

Gruppierung logischer Knoten

Die Vorbedingung für die Gruppierung logischer Knoten ist, dass alle Blätter der logischen Baumstruktur durch Gruppierung von physischen Knoten erzeugt sind. Es folgt zunächst eine formale und anschliessend eine verbale Darstellung

Dies ist nur bei einem Knoten möglich, der ein *letzter* Nachfolgeknoten eines physischen Knotens ist (siehe Definition der Trigramme i^f, i^l in Kapitel 4), was erst für den letzten Knoten der b12 Folge, als letzter Nachfolger von ll zutrifft.

des Algorithmus. Die Ziffern, die in beiden Darstellungen den Algorithmusblöcken vorangestellt sind, entsprechen einander. Der Algorithmus beschreibt einen einzelnen Gruppierungsschritt, in dem eine Gruppe von logischen Knoten zu einem logischen Unterbaum zusammengefasst wird.

Definitionen:

- $anc(\lambda)$: die Menge aller möglichen Vorgänger (ancestors) einer logischen Entität λ.
$$anc : \Lambda \rightarrow \mathcal{P}(\Lambda)$$
$$anc(\lambda) = \{\mu \mid a_{\lambda\mu} > 0\}$$

- $depth(\lambda)$: die minimale Tiefe (von der Wurzel aus gesehen), in der sich λ in einem vollständigen logischen Baum befinden kann.
$$depth(\lambda) = \begin{cases} 0, \text{ if } a_{\lambda\lambda_0} > 0 \\ \min_{\mu\in\Omega}(1 + depth(\mu)), \quad \Omega = \{\mu \mid a_{\lambda\mu} > 0\} \end{cases}$$

- $sequence(S, \mu)$: ist TRUE, wenn λ der rechte Nachbar irgendeines Elementes aus S sein kann, das heisst, wenn sie eine Sequenz bilden können.
$$sequence(S, \mu) = \begin{cases} \text{TRUE}, & \text{if } \exists\lambda \in S, r_{\lambda\mu} > 0 \\ \text{FALSE}, & \text{else} \end{cases}$$
mit $S \subseteq \Lambda$ und $\mu \in \Lambda$.

Algorithmus: (siehe auch Abbildung 6.5)

1. Alle logischen Knoten einer partiellen Lösung, die entweder Blätter oder Wurzeln von Teilbäumen sind, bilden in Lesefolge eine Sequenz, dargestellt durch das Argument log-nodes. Mit der Referenz q wird die Sequenz beginnend mit dem ersten Element von log-nodes durchlaufen.

2. Berechne maxdepth, das die aktuelle Baumtiefe im logischen Baum bezeichnet, in der mit den nächsten Schritten logische Knoten gruppiert werden. Dieser Wert wird benötigt, um die bottom-up Konstruktion der logischen Struktur bei den am tiefsten liegenden Knoten fortsetzen zu können.

3. Suche nach einem Kandidaten für den ersten logischen Knoten einer Gruppe. Rücke in der Sequenz q solange vor, bis folgendes Abbruchkriterium erfüllt ist:

```
TYPE  lnode:
            label: Λ
            ancestor: lnode
      sequence:
            content: lnode
            next: sequence

ARG   log-nodes: sequence

VAR   q: sequence
      first, last: lnode
      κ: Λ
      common-anc: P(Λ)
      maxdepth: N0

BEGIN
1.    q ← log-nodes
2.    maxdepth ← max_{∀n in q}( depth(n.label)),

3.    while ¬( depth(q.content)=maxdepth
               ∧ (l_{q.content,λ0} = 1
                  ∨ (sequence( anc(q.prev.label, q.content.label)))))
            q ← q.next
      first ← q.content

      common-anc ← anc(q.content.label)
4.    while ¬(common-anc = ∅
               ∨(r_{q.content.label,λ0} > 0
                  ∧(l_{q.next.label,λ0} > 0
                     ∨ common-anc ∩anc(q.next.label) = ∅)
                  ∧r_{q.content.label,q.next.label} = 0))
            q ← q.next
            common-anc ← common-anc ∩anc(q.content.label)
      last ← q.content

5.    foreach κ ∈ common-anc
            „group first...last to logical entity with label κ“
END
```

Abbildung 6.5: Algorithmus für die Gruppierung logischer Knoten.

(a) q.content liegt auf einem genügend tiefen Niveau (*depth*(q.content.label) = maxdepth), um die bottom-up Konstruktion des logischen Baums zu garantieren, und

(b) entweder muss q.content ein erstes Element sein ($l_{\text{q.content.label},\lambda_0} = 1$) oder kein Vorgängerknoten von q.prev kann ein linker Nachbar von q.content sein.

Ein Kandidat für den ersten zu gruppierenden logischen Knoten wird durch first repräsentiert.

4. Suche nach einem Kandidaten für den letzten logischen Knoten der Gruppe. Rücke in der Sequenz q solange vor, bis folgendes Abbruchkriterium erfüllt ist: entweder die Sequenz von logischen Knoten von first bis q.content hat keine gemeinsamen Vorgängerknoten (common-anc $\neq \emptyset$), oder

   (a) q.content kann ein letzter Knoten sein ($r_{\text{q.content.label},\lambda_0} > 0$), und

   (b) der nächste Knoten q.next kann den Anfang einer neuen Gruppe bilden $l_{\text{q.next.label},\lambda_0} > 0$, oder die aktuelle Gruppe wird zusammen mit der nächsten Entität keine gemeinsamen Vorgängerknoten haben können (common-anc $\cap anc$(q.next.label) $= \emptyset$)), und

   (c) der aktuelle Knoten kann mit dem Folgeknoten keine Sequenz bilden ($r_{\text{q.content.label,q.next.label}} = 0$).

   Ein Kandidat für den letzten zu gruppierenden logischen Knoten wird durch last repräsentiert.

5. Fasse die logischen Knoten von first bis last in einen Unterbaum mit der logischen Entität κ als Wurzel zusammen. Die Alternativen für κ entsprechen der Menge der gemeinsam möglichen logischen Vorgänger für die Gruppe von logischen Knoten.

In Schritt 5. werden analog zu der Gruppierung physischer Knoten entweder keine, eine oder mehrere neue partielle Lösungen erzeugt.

Betrachten wir als Beispiel einer Gruppierung logischer Knoten die Abbildung 6.6. Die Sequenz q enthält die Wurzeln aller Unterbäume einer partiellen Lösung SECT, SECT, STIT, P, ... Die Ziffer in Klammern, neben dem logischen Label in der Abbildung bezeichnet die minimale Tiefe (*depth*) jeder logischen Entität. maxdepth beträgt folglich 2 in unserem Beispiel. Auf der Suche nach einem ersten Knoten für die Gruppierung werden die beiden ersten SECT Knoten nicht in Betracht gezogen, weil sie sich nicht in der aktuellen Konstruktionstiefe

Abkürzungen:	Section	SECT	
	Section Title	STIT	
	Paragraph	P	
Modell:	$a_{\mathsf{STIT,SECT}} = 1$	$l_{\mathsf{STIT},0} = 1$	$r_{\mathsf{P,P}} = 0.6$
	$a_{\mathsf{P,SECT}} = 0.2$		$r_{\mathsf{P},0} = 0.2$
	$a_{\mathsf{P,SECT1}} = 0.45$		$r_{\mathsf{P,STIT}} = 0$
	$a_{\mathsf{P,SECT2}} = 0.40$		$r_{\mathsf{STIT},0} = 0$

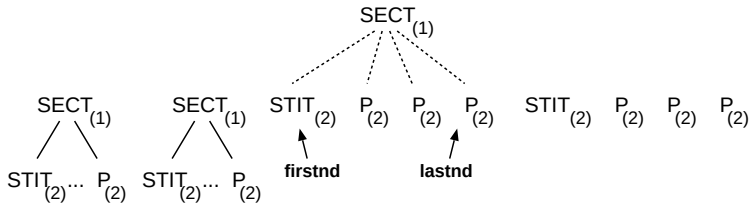


Abbildung 6.6: Beispiel einer Gruppierung von logischen Knoten.

2 befinden können. Erst STIT ist ein Kandidat für einen ersten Knoten, weil er ein erster Knoten eines Unterbaumens sein muss ($l_{\mathsf{STIT},0} = 1$).

Die Menge common-anc aller möglichen Vorgänger von STIT enthält das Element SECT, weil $a_{\mathsf{STIT,SECT}} = 1$. Jetzt wird nach einem letzten Knoten der Gruppe gesucht: die beiden ersten P Knoten kommen nicht in Frage, weil eine Sequenz mit dem auf sie folgenden Knoten P möglich ist ($r_{\mathsf{P,P}} > 0$). Der letzte P Knoten ist hingegen ein Kandidat für den letzten Knoten der Gruppe, weil P mit STIT keine Sequenz bilden kann, und weil STIT ein möglicher erster Unterknoten eines Unterbaumes ist. Die Schnittmenge der möglichen Vorgängerknoten von STIT bis P ist $\{\mathsf{SECT}\} \cap \{\mathsf{SECT}, \mathsf{SECT2}, \mathsf{SECT2}\} = \{\mathsf{SECT}\}$. Schliesslich werden die Knoten first, ..., last aus q zu einem Unterbaum mit dem logischen Label SECT als Wurzel zusammengefasst.

6.1.4 Suche der optimalen Lösung

Wie wir bisher gesehen haben, lässt sich die logische Struktur nicht in einer eindeutigen Konstruktionsreihenfolge aufbauen. Aus manchen Konstruktionsschritten resultieren konkurrierende Alternativen. Dieser Effekt ist durchaus

erwünscht, ermöglicht er es doch, trotz lokal entschiedener Konstruktionsschritte eine global valide Lösung zu finden.

Für die Suche nach einer optimalen Lösung verwenden wir eine best-first Suchstrategie [Win84, Pea85b], deren Prinzip in Abbildung 6.7 illustriert ist. Die Abbildung zeigt einen Ausschnitt aus dem *Suchbaum*, von dem Zeitpunkt an, bei dem eine partielle Lösung π_i generiert wurde. Der Algorithmus für die Knotengruppierung kann zwar eine Gruppe von Knoten finden, die zu einem Unterbaum zusammengefasst werden sollen, er kann aber die Wurzel des Unterbaums nicht eindeutig bestimmen. Es ergeben sich zwei konkurrierende Hypothesen H_1 und H_2 für das Label der Wurzel des neuen Unterbaums. Die best-first Suchstrategie schreibt nun vor, beide Alternativen zu realisieren und π_{i+1} und π_{i+2} zu generieren. Beide Alternativen werden anschliessend einer Bewertung unterzogen, in der ihre *Qualität* gemessen wird. Mit dem Begriff der Qualität ist die Plausibilität gemeint, dass eine partielle Lösung π_k Teil einer global optimalen Lösung ist. In unserer Anwendung entspricht die Qualität der Konformität einer partiellen Lösung mit dem Dokumentmodell. Das Mass dafür ist die Generalized-Means-Funktion h_α.

Die Generalized-Means-Funktion wird hier in exakt derselben Weise verwendet, wie in Kapitel 4 für die Konformitätsmessung zweidimensionaler Strukturen, die mit generalisierten n-Grammen modelliert sind. Das Prinzip besteht darin, in einer partiellen Lösung π die lokale Kohärenz sämtlicher Nachbarschaftsbeziehungen durch die entsprechenden n-Gramm-Wahrscheinlichkeiten auszudrücken, um sie mit der Generalized-Means-Funktion zu einem einzigen Wert zusammenzufassen. Augedrückt als Pseudoformalismus ist die Generalized-Means-Funktion für eine partielle Lösung π definiert als:

$$h_\alpha(\pi) = \left(\frac{1}{|\Gamma_\pi|} (\mathbf{x}_{\lambda\mu\nu})^\alpha \right)^{\frac{1}{\alpha}}$$

wobei

$$\Gamma_\pi = \{\text{„Menge aller Nachbarschaftsbeziehungen in } \pi\text{“}\}$$
$$\mathbf{x} \in \{r, l, t, b, e, i^f, i^l, h^f, h^l\}.$$

Für das Beispiel in Abbildung 6.7 sei angenommen, dass die Beziehung $h_\alpha(\pi_{i+1}) > h_\alpha(\pi_{i+2})$ gilt. Da π_{i+1} somit als aussichtsreichere partielle Lösung gilt, wird die weitere Konstruktion an dieser Stelle fortgeführt. Es ist für den weiteren Verlauf folgendes Szenario denkbar: Aus π_{i+1} folgen die zwei Alternativen π_{i+3} und π_{i+4} mit $h_\alpha(\pi_{i+3}) < h_\alpha(\pi_{i+4})$. In π_{i+4} lässt sich aber keine mögliche Gruppierung finden (Die Suche nach einer Gruppe kann ja auch erfolglos ausgehen). Die Konstruktion wird bei π_{i+3} fortgeführt, wobei π_{i+5} erzeugt wird. Wenn wir annehmen, dass $h_\alpha(\pi_{i+5}) < h_\alpha(\pi_{i+2})$, dann wird die Konstruktion bei π_{i+2} fortgesetzt. Der gesamte Ast im Suchbaum unterhalb

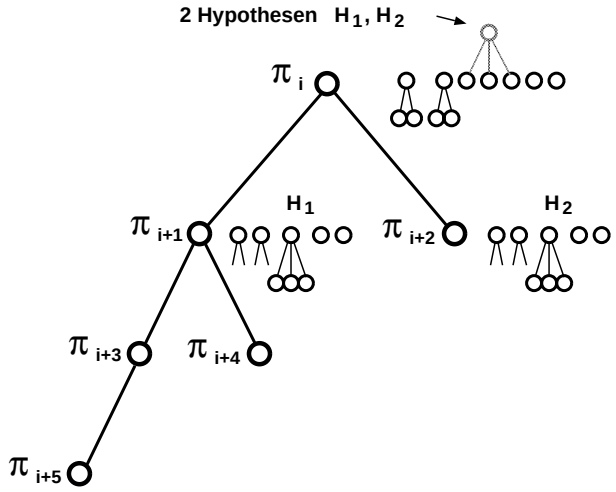


Abbildung 6.7: Teil des Suchbaums beginnend mit der partiellen Lösung π_i.

von π_{i+1}, beziehungsweise π_{i+5}, wird erst wieder in Erwägung gezogen, wenn $h_\alpha(\pi_{i+5})$, verglichen mit allen anderen partiellen Lösungen, den grössten Wert erreicht.

Der Grund, warum der Ast im Suchbaum unterhalb von π_{i+1} verlassen wird, obwohl π_{i+1} im Vergleich zu π_{i+2} besser zu bewerten ist, liegt daran, dass lokale Kriterien den Ausschlag für die bessere Bewertung von π_{i+1} gegeben haben. Erst später, wenn mit π_{i+5} ein grösserer Kontext bekannt ist, stellt sich heraus, dass mit der Hypothese H_1 vorläufig keine optimale Lösung zu erreichen ist.

6.2 Optimierungen

Der im vorangegangenen Abschnitt vorgestellte Algorithmus für die Erkennung, beziehungsweise Konstruktion der logischen Struktur, ist bereits voll funktionstüchtig. Vorversuche haben aber gezeigt, dass gewisse Optimierungen nötig sind, damit der Algorithmus auch in reellen Situationen befriedigende Erkennungsleistungen erbringt.

Das Kernproblem ist die heuristische best-first-Suche nach einer optimalen Lösung. Man kann sich leicht vergegenwärtigen, dass für Dokumente mit einigen tausend Wörtern der Suchbaum enorme Ausmasse annimmt. Die anschliessend vorgestellten drei Optimierungen haben alle zum Ziel, den Suchbaum möglichst

klein zu halten, ohne jedoch das Grundprinzip des Ansatzes zu verändern.

6.2.1 Erweitertes Konformitätsmass

Die Generalized-Means-Funktion h_α, als Mass für die Modellkonformität von partiellen Lösungen, hat in Vorversuchen ihren Zweck gut erfüllt. Insbesondere eignet sie sich für den Vergleich von partiellen Lösungen, der ja auch ihr Hauptzweck in der best-first Suche ist. Die Bewertungsfunktion spielt eine zentrale Rolle in der Suche, da ihre Werte die alleinigen Einflussfaktoren auf die Art und Weise sind, wie der Suchbaum aufgebaut und durchlaufen wird. Es hat sich nun gezeigt, dass mit der Generalized-Means-Funktion der Suchbaum tendenziell zu breit wird. Wir fügen daher einen zusätzlichen Term in die Bewertungsfunktion ein, mit dem sie sich so steuern lässt, dass der Suchbaum schneller in die Tiefe wächst:

$$h'_\alpha(\pi_i) = h_\alpha(\pi_i)|\pi_i|^s$$

$|\pi_i|$ bezeichnet die Anzahl logischer Knoten in der partiellen Lösung π_i, also ihre „Grösse". Der Parameter s steuert die Einflussstärke der Grösse von π_i. Wenn $s = 0$ beträgt, hat die Grösse von π_i keinen Einfluss, und die Suchstrategie verhält sich ähnlich zu einer breadth-first-Suche. Wird für s ein hoher Wert gewählt, zum Beispiel $s = 5$, dominiert die Grösse von π_i fast vollständig den Wert von h'_α, und die Suchstrategie nähert sich einer depth-first Suche an. In der Praxis haben sich für s Werte zwischen 0 und 1 bewährt.

6.2.2 Aufteilung in Teilprobleme

Die strikte bottom-up Konstruktionsmethode der logischen Struktur kann dazu führen, dass erstens falsch gewählte Hypothesen zu spät erkannt werden, und dass zweitens eine ganze Reihe korrekt konstruierter Knoten verworfen und neu konstruiert werden muss.

Die Abbildung 6.8 zeigt ein solches Szenario. Die eingekreisten Ziffern geben die Konstruktionsreihenfolge an. Wir nehmen an, dass für die Gruppierung bei ① zwei Hypothesen für das logische Label generiert wurden, und dass aufgrund der lokalen Bewertung die falsche Hypothese weiterverfolgt wird. Wir nehmen ferner an, dass die Konstruktionschritte bei ②, ③ und ④ korrekt vollzogen werden. Erst bei Schritt ⑤ kann der Algorithmus nun feststellen, dass bei ① die falsche Hypothese gewählt wurde, weil erst dann genügend Kontextinformation zur Verfügung steht. Die Backtracking-Strategie erfordert nun, dass alle Konstruktionsschritte von ④ bis ② rückgängig gemacht werden, um die Alternativhypothese bei ① realisieren zu können. Der Algorithmus wird in der Folge die exakt gleichen Schritte ② bis ④ durchführen wie zuvor, um schliesslich bei

⑤ eine korrekte Gruppe zu bilden. Diese offensichtliche Ineffizienz ist besonders gravierend, wenn bei ④ eine grosse Zahl von Knoten gebildet wurde.

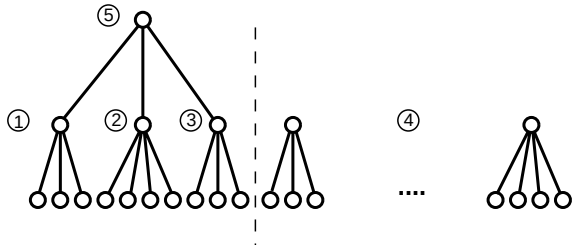


Abbildung 6.8: Problemszenario und Aufsplittung in Teilprobleme.

Die Ineffizienz lässt sich erheblich verringern durch eine Aufteilung des Problems in Teilprobleme, die getrennt behandelt werden. Wird, wie in Abbildung 6.8 gezeigt, das Problem bei der vertikalen Linie geteilt, so müssen die Schritte bei ④ nicht realisiert werden, um den Fehler bei ① zu detektieren.

Als Stellen für die Aufteilung des Problems kommen physische Entitäten in Frage, die mit hoher Sicherheit das Problem in Teilprobleme aufteilen. In der Regel existieren eine Reihe solcher physischen Entitäten. So ist beispielsweise sehr gross und fett geschriebener Text nur in Kapitelüberschriften zu finden und eignet sich daher für die Aufteilung eines Artikels in Kapitel.

Allgemein eignen sich all jene Stellen mit physischen Entitäten φ_i, wenn Trigramme $i^f_{ijk} \doteq 1$ existieren, wobei λ_k eine logische Entität ist, die sich nicht tief im Baum befindet (z.B. $depth(\Lambda_k) = 0, 1$ oder 2).

Wenn nach einer Aufteilung die Teilprobleme gelöst sind, können diese wieder zu einem Problem zusammengefasst und auf die übliche Art zu Ende konstruiert werden.

6.2.3 Schätzung der Hypothesenqualität

Die Realisierung einer Hypothese, also die Generierung einer neuen partiellen Lösung π_{i+1} aus π_i, ist sehr aufwendig. Sämtliche Teilbäume der partiellen Lösung π_i müssen kopiert werden, wobei die Knotenzahlen bei praktischen Problemen schnell in die Tausenden gehen. Es lohnt sich also, schon vor der Realisierung einer Serie von Hypothesen deren Wert abzuschätzen, um die vielversprechenden vorrangig zu realisieren.

Die Grundidee ist nun, vor der Gruppierung einer Reihe von Knoten einige der neu hinzukommenden n-Gramm-Wahrscheinlichkeiten herauszugreifen, um daraus eine Schätzung für die Erfolgsaussicht einer Gruppierungsoperation zu

berechnen. Hier eignen sich die Bigramme $a_{\text{first.label},\mu}$ und $a_{\text{last.label},\mu}$ für eine Erfolgsschätzung der Gruppierung von first bis last mit μ als Wurzel.

Bezogen auf das Beispiel in Abbildung 6.7 würde das bedeuten, dass nach π_i zunächst nur π_{i+1} generiert wird, nicht aber π_{i+2}. Erst nachdem die Versuche mit π_{i+3}, π_{i+4} und π_{i+5} gescheitert sind, wird π_{i+2} realisiert. Je nach Breite des Suchbaumes bringt diese verzögerte Konstruktion von partiellen Lösungen eine erhebliche Ersparnis an verbrauchtem Speicherplatz.

6.3 Diskussion

In den folgenden Abschnitten sollen einige wichtige Eigenschaften des Algorithmus für die Konstruktion der logischen Struktur diskutiert werden. Es werden die nötigen Parameter, die Beschränkungen des Ansatzes, die Fehlertoleranz und die Komplexität des Algorithmus näher betrachtet.

6.3.1 Parameter

Für die Erkennung der logischen Struktur müssen insgesamt drei Parameter festgelegt werden.

α: Der Parameter α beeinflusst die Generalized-Means-Funktion, und diese beeinflusst wiederum die Weise, wie die logische Struktur aufgebaut wird, und welche partiellen Lösungen zurückgewiesen werden. Der in der Praxis sinnvolle Bereich liegt bei $0 \leq \alpha \leq 1$.

Ist α nahe bei 0, reagiert die Bewertung sehr empfindlich auf n-Gramm-Wahrscheinlichkeiten, die nahe bei 0 liegen. Der Algorithmus wird daher intolerant gegenüber Fehlern in der logischen Struktur. Liegt α nahe bei 1, vermögen einzelne Fehler die Gesamtwertung nur schwach zu beeinflussen. Der Algorithmus wird dann tolerant gegenüber Fehlern in der logischen Struktur.

s: Der zweite Parameter, der die Bewertung von partiellen Lösungen beeinflusst, ist der Grössenexponent s. Er bestimmt, wie stark die Grösse einer partiellen Lösung, das heisst, die Anzahl ihrer logischen Knoten, in die Gesamtbewertung einfliesst. Wenn $s = 0$ ist, hat die Grösse keinen Einfluss auf die Bewertung. Durch die Charakteristik der Generalized-Means-Funktion h_α ist die Suche dann einer breadth-first Suche ähnlich. Schon bei Werten für s um 3 oder 4 jedoch überdeckt die Grösse einer partiellen Lösung fast vollständig h_α. Der best-first Algorithmus wird infolgedessen die grössten partiellen Lösungen bevorzugt weiterentwickeln. Da die Grösse von partiellen Lösungen direkt ihrer Tiefe im Suchbaum

entspricht, gleicht sich die Suche einer depth-first Strategie an. Sinnvolle Werte liegen im Bereich $0 \leq s \leq 1$.

splitdepth: Dieser Parameter ist kein Skalar, sondern ein ganzzahliges Intervall und wurde noch nicht explizit eingeführt. *splitdepth* beeinflusst die Aufteilung eines Erkennungsproblems in Teilprobleme. Um Teilungsstellen ausfindig zu machen, sucht der Algorithmus nach logischen Entitäten, die erstens ein sehr charakteristisches Layout aufweisen, und zweitens sich nicht zu tief in der logischen Baumstruktur befinden, damit die Teilprobleme nicht zu klein werden. Die Tiefe, in der sich die logische Entität befindet, wird von *splitdepth* bestimmt. Hierfür lassen sich keine generell sinnvollen Werte angeben, weil sie stark von der Struktur einer Dokumentklasse abhängen können. In unseren Tests wurde $splitdepth = [1, 2]$ festgesetzt.

Für den Anwender eines Programms ist es in der Regel angenehm, wenn er zu Beginn dessen Gebrauchs nicht eine Serie von Parametern bestimmen muss. Oftmals ist für die Festlegung von Parametern auch Expertenwissen nötig, was den Kreis von Anwendern stark einschränkt. Weitgehende Parameterfreiheit sollte deshalb für Anwendungen wie die Dokumenterkennung ein Ziel sein.

Unsere Tests haben gezeigt, dass der Erkennungsalgorithmus auf Variationen der Parameter α und s unempfindlich reagiert. Daher mussten deren Werte praktisch nie verändert werden. Etwas problematischer ist der Parameter *splitdepth*. Hier ist es offensichtlich, dass er an die zu erkennende Dokumentklasse angepasst werden muss. Allerdings ist ein Einblick in die Funktionsweise des Algorithmus nötig, um ihn festlegen zu können. Die Möglichkeit einer automatischen Festlegung des Parameters ist hingegen nicht ausgeschlossen.

6.3.2 Beschränkungen

Der Erkennungsalgorithmus besitzt von Grund auf einige Beschränkungen, die die Erkennung gewisser Dokumentklassen ausschliessen.

Eine Beschränkung des Algorithmus ist das Verbot rekursiver Dokumentstrukturen. Es ist also nicht erlaubt, dass logische Entitäten sich selbst enthalten. Der Grund dafür liegt in subtilen Eigenschaften des Erkennungsalgorithmus, deren Erläuterung hier zu weit führen würde. Da das Verbot rekursiver Strukturen aber nicht im Zusammenhang mit der Expressivität des Modells steht, sind alternative Erkennungsalgorithmen denkbar, die Rekursionen in der Struktur zulassen.

Eine weitere Beschränkung ist der Ausschluss zweidimensionaler Strukturen. Der Algorithmus ist darauf ausgelegt, dass eine klar definierte Leseordnung existiert: eine Ordnung von Wörtern in einer Zeile, von Zeilen in einem Block, von

Blöcken in einer Seite und von Seiten im Dokument. Nur Dokumente mit dieser Eigenschaft der definierten Leseordnung lassen sich adäquat als Baumstruktur darstellen. Typische zweidimensionale Strukturen, wie Tabellen oder mathematische Formeln sind mit dem vorgeschlagenen Modell nicht repräsentierbar.

Schliesslich wurden weder der Algorithmus noch das Modell darauf ausgelegt, sogenannte *floating bodies* zu erkennen. Floating bodies sind meist grössere Blöcke, die im Layout eine gössere, von der Leseordnung unabhängige Platzierungsfreiheit haben als normale Textelemente. Typische floating bodies sind Graphiken, Bilder und Tabellen. Glücklicherweise haben floating bodies im allgemeinen sehr charakteristische graphische Eigenschaften und können daher leicht bei der Erkennung der physischen Struktur herausgefiltert werden.

6.3.3 Komplexität

Die Konstruktion der logischen Struktur ist ein *state-space-search* Problem, das heisst, in einem Raum von diskreten Zuständen suchen wir einen bestimmten Zustand. Nehmen wir vereinfachend an, dass unser Problem darin besteht, auf n gegebenen Blättern eine bestimmte Baumstruktur aufzubauen. Nun ist die Anzahl der verschiedenen Bäume mit denselben n Blättern unendlich. Selbst wenn wir einschränken, dass ein Knoten mindestens zwei Nachfolger haben muss (eine Einschränkung, die für Dokumentstrukturen oft, aber nicht immer gilt), wächst die Anzahl möglicher Bäume exponentiell mit n an.

Nun trägt unser Erkennungsalgorithmus diesem Umstand aber Rechnung, indem die Baumstruktur so gezielt wie möglich konstruiert wird. Er generiert mittels komplexer Regeln nur sehr aussichtsreiche Hypothesen darüber, wie die Struktur um einen Knoten erweitert wird. In sehr vielen Fällen wird sogar nur eine einzige Hypothese vorgeschlagen. Dies führt dazu, dass die exponentielle Komplexität des Algorithmus in der Praxis kaum spürbar ist. In der Evaluation in Kapitel 7 wird gezeigt, dass die Anzahl der durchgeführten Konstruktionsschritte selten die doppelte Anzahl der Knoten in der Lösung übersteigt.

6.4 Alternative Erkennungsansätze

Nach der Präsentation des Konstruktionsalgorithmus stellt sich die Frage, ob sich in der Literatur Ansätze finden lassen, die ähnliches zu leisten imstande sind, und somit als Alternative oder Ergänzung zu unserem Algorithmus eingesetzt werden könnten.

Betrachten wir noch einmal die abstrakten Eigenschaften unserer Problemstellung. Gegeben ist ein Baum (die physische Struktur), dessen Knoten mit Labels attribuiert sind. Gesucht ist ein zweiter Baum (die logische Struktur), der unter Berücksichtigung komplexer Regeln zum ersten Baum passt. Die Knoten

des gesuchten Baumes sind ebenfalls mit Labels attribuiert. Zudem sollen auch die Verbindungen zwischen den beiden Baumstrukturen hergestellt werden.

Beim Studium der einschlägigen Literatur sind wir auf kein bekanntes Verfahren gestossen, das die obigen Anforderungen erfüllt. Auch ist uns keine Möglichkeit bekannt, wie das Problem zurückgeführt werden könnte auf Probleme mit gut dokumentierten Lösungsansätzen, wie zum Beispiel Klassifikation, graph-matching oder constraint-based reasoning. Der Grund dafür, warum unser Dokumenterkennungsproblem nicht auf ein bekanntes Problem zurückzuführen ist, liegt in seiner Formulierung. Die Konstruktion und das Aufsetzen einer Baumstruktur auf eine gegebene Baumstruktur beinhaltet eine Segmentierung (die Gruppierung von physischen und logischen Entitäten) und eine Klassifizierung (Finden des Labels eines neuen Knotens) und zwar unter Berücksichtigung des benachbarten Kontexts. Allerdings sind die Phasen der Segmentierung und der Klassifizierung so stark gegenseitig abhängig, dass sie nicht sequentiell abgearbeitet werden können.

Grundsätzlich ist festzuhalten, dass die meisten Ansätze aus dem Bereich der künstlichen Intelligenz und machine learning als Resultat im wesentlichen eine einzelne isolierte Grösse liefern, zum Beispiel eine Entscheidung, eine Klasse oder eine Diagnose. Verfahren hingegen, die komplette Strukturen als Ergebnis liefern können, sind selten. Eine wichtige Ausnahme bildet die Klasse der syntaktischen Ansätze. Sie erzeugen bei der Analyse einer linearen Struktur, beispielsweise einer Zeichenkette, wiederum eine Struktur: nämlich einen Parsingbaum. Es ist allerdings unklar, wie eine Baumstruktur anstelle einer linearen Struktur mit einer syntaktischen Methode analysiert werden kann, so dass der erzeugte Parsingbaum genau der gesuchten Baumstruktur entspricht.

6.5 Zusammenfassung

Für die Erkennung der logischen Struktur eines Dokuments, die sich auf ein zuvor gelerntes n-Gramm-Modell abstützt, eignet sich besonders ein heuristisches Verfahren. Heuristiken bieten sich an, wenn unscharfe Daten verarbeitet werden sollen, und wenn anstelle eines formalen Verfahrens lediglich ad hoc Regeln und auf dem gesunden Menschenverstand basierende Vorgehensweisen bekannt sind.

Der Ansatz besteht in einer schrittweisen bottom-up Konstruktion der logischen Struktur, die auf die physische Struktur aufgesetzt wird. Aufgrund von Regeln, die die physische und die bereits konstruierte logische Struktur sowie das Modell interpretieren, werden Hypothesen darüber erzeugt, wie sich in einem nächsten Schritt ein neuer Knoten an die Struktur (partielle Lösung) anfügt. Verschiedene Alternativen von partiellen Lösungen werden mit der Generalized-Means-Funktion im Hinblick auf ihre Modellkonformität laufend bewertet. Eine

optimale Lösung wird mit einer best-first Suchstrategie gefunden. Optimierungen sorgen dafür, dass die theoretisch exponentielle Komplexität des Algorithmus in der Praxis nicht ins Gewicht fällt.

Kapitel 7

Evaluation

In diesem Kapitel wird mit empirischen Tests der Nachweis der Tauglichkeit unseres Ansatzes für die Modellierung und Erkennung von Dokumentstrukturen erbracht. Die Tests haben zum Ziel, die folgenden Fragen zu klären.

1. Wie gut ist das Modell in der Lage, Dokumentstrukturen zu repräsentieren?

2. Kann der Erkennungsalgorithmus mit Hilfe des Modells die logische Struktur von Dokumenten konstruieren?

3. Wie stark vermögen inkrementelle Lernschritte das Modell mit der Zeit zu verbessern?

4. Ist mit der Zeit eine Konvergenz des Modells festzustellen?

5. Wie effizient ist der Erkennungsalgorithmus?

Zunächst werden in Abschnitt 7.1 einige Vorversuche und deren Resultate vorgestellt. Sie basieren auf dem Modell und dem Erkennungsalgorithmus in einem früheren Entwicklungsstadium, als sich insbesondere der Algorithmus vom derzeit aktuellen unterschieden hatte. Anschliessend betrachten wir die in der Hauptversuchsreihe verwendeten Testdaten (Abschnitt 7.2), deren Aufbereitung (Abschnitt 7.3) und die Ergebnisse der Versuchsdurchgänge (Abschnitt 7.4). Das Kapitel schliesst mit einer eingehenden Diskussion der Resultate in Abschnitt 7.6.

7.1 Vorversuche

Der bisher vorgestellte Ansatz zur Modellierung und Erkennung von Dokumentstrukturen ergibt sich aus einer langen Reihe von Entwürfen und Versuchen. Die Grundidee der statistischen Repräsentation mit n-Grammen bestand von Anbeginn der Forschungen und musste nur marginal erweitert werden. Die meisten Veränderungen erfuhr im Laufe der Zeit der Erkennungsalgorithmus, der kontinuierlich weiterentwickelt wurde. Dennoch ist ein Rückblick auf frühere Resultate interessant, um die Stärken und Schwächen des Ansatzes zu verdeutlichen.

Wir betrachten die Versuchsreihe „Memento“ für die Erkennung eines Veranstaltungskalenders sowie die Reihe „Medical Imaging“ für die Erkennung von wissenschaftlichen Publikationen.

7.1.1 „Memento“ Dokumente

„Memento“ Dokumente sind einseitige Veranstaltungskalender, die wöchentlich intern an der Universität Freiburg erscheinen, und eignen sich gut für die Erkennung, da sie eine reichhaltige Struktur aufweisen, die sich stark im Layout widerspiegelt. Die Abbildung 7.1 zeigt links das Bild eines „Memento“ Dokuments und rechts einen vergrösserten Ausschnitt, während Abbildung 7.2 einen Ausschnitt aus der logischer Struktur zeigt. Es existieren zwei Typen von Veranstaltungen: Konferenzen, die den Sprecher, seine Herkunft, den Titel und Zusatzinformationen umfassen sowie Versammlungen, die lediglich den Titel und Zusatzinformationen aufführen. Optional endet ein Veranstaltungseintrag mit einer Referenz.

Im Unterschied zum Modell, wie es in Kapitel 4 definiert wurde, existiert keine Unterscheidung zwischen einer physischen und logischen Strukturebene. Das Modell enthält daher nur die n-Gramme r, l, t, b und e. Wie die Abbildung 7.1 zeigt, entspricht der Strukturbaum der logischen Struktur, wobei sich allerdings die Blätter auf physische Entitäten beziehen. Eine einzelne physische Entität entspricht einer Textzeile, und sie ist nach dem in der Zeile vorwiegend verwendeten Zeichensatz klassifiziert. Insgesamt existieren fünf Klassen physischer Entitäten: HEL12BI, HEL14I, HEL12, HEL10, AG14 (für Helvetica 14pt bold italic, 14pt italic, 12pt, 10pt und Avant Garde 14pt).

Der Erkennungsalgorithmus konstruiert bottom-up die logische Struktur auf den gegebenen physischen Blättern. Bei jedem Schritt wird entweder ein neuer Vorgängerknoten für einen einzelnen Baumknoten erzeugt, oder ein bestehender Knoten wird in einen Unterbaum integriert. Im Unterschied zum Algorithmus in Kapitel 6 wird also nicht in einem Zug eine Gruppe von Knoten zu einem Unterbaum zusammengefasst, sondern es wird schrittweise immer nur eine Ver-

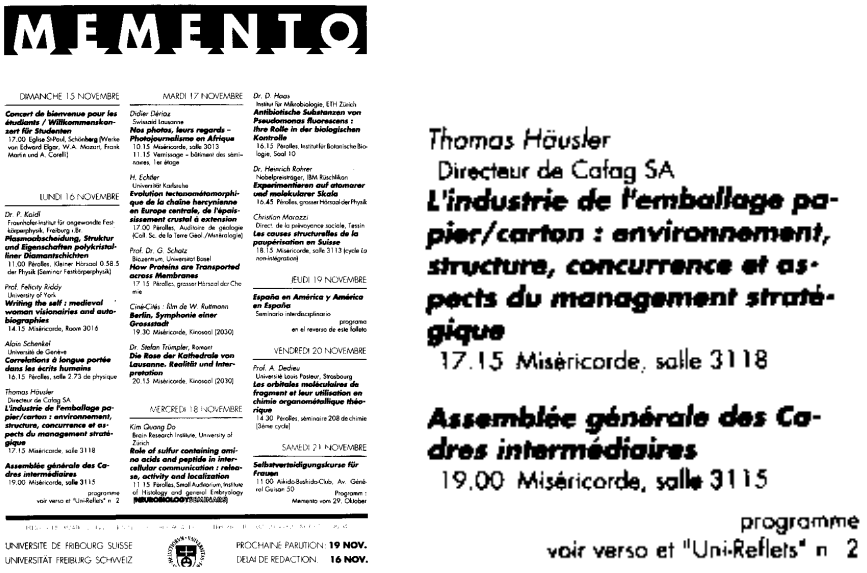


Abbildung 7.1: Eine typische Seite eines „Memento“ Dokuments und ein vergrösserter Ausschnitt.

bindung zwischen den Knoten aufgebaut. Die Erkennung besteht somit darin, auf einer Sequenz von physischen Entitäten die logische Struktur aufzubauen. Die Sequenz von physischen Entitäten wurde in unseren Versuchen von Hand erstellt, und ist daher frei von Fehlern.

gelernte Einträge	erkannte Einträge	nicht erkannte Einträge
2	13	5 (28%)
3	15	2 (17%)
4	18	0 (0%)

Tabelle 7.1: Resultate der Erkennung einzelner Veranstaltungen.

In Tabelle 7.1 sind die Resultate aufgelistet, die sich aus der Erkennung einzelner Veranstaltungseinträge ergeben haben. Werden zwei Einträge gelernt (eine Konferenz und eine Versammlung), so können damit 13 Veranstaltungen einer Dokumentseite erkannt werden. Durch Hinzulernen einer weiteren Veran-

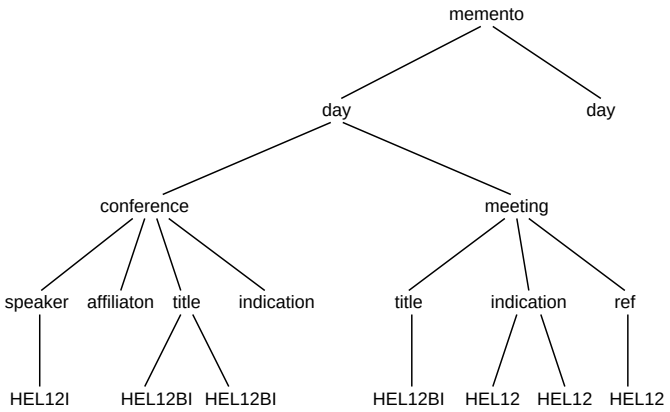


Abbildung 7.2: Ausschnitt aus der logischen Struktur eines typischen „Memento“ Dokuments.

staltung werden 15 korrekt erkannt, und nach einem weiteren Lernschritt wird eine hundertprozentige Erkennungsrate erreicht.

In einer weiteren Versuchsanordnung wird die Erkennung kompletter, einseitiger Dokumente erprobt. Es hat sich dabei herausgestellt, dass mit einer gelernten Seite fünf weitere vollständig und korrekt erkannt werden.

Aus den Resultaten kann geschlossen werden, dass das n-Gramm-Modell Strukturen darstellen und korrekte von fehlerhaften Strukturen unterscheiden kann, sowie dass das Modell vom Erkennungsalgorithmus für den gezielten Strukturaufbau verwendbar ist. Nur wenige Lerndurchgänge sind nötig, um ein brauchbares Modell zu lernen. Relativierend ist allerdings festzuhalten, dass „Memento“ Dokumente eine einfache Struktur aufweisen, die sich oft direkt im Layout widerspiegelt. Dadurch, dass die Konstruktion der Struktur vergleichsweise langsam abläuft und bei vielen Konstruktionsschritten mehrere Alternativen zur Auswahl stehen, besteht zudem die Gefahr, dass der Suchbaum zu gross und die Suche nach Lösungen ineffizient werden kann. Die Grösse der „Memento“ Dokumente liegt jedoch in einem Bereich, der gut zu bewältigen ist.

7.1.2 „Medical Imaging“ Artikel

In einer weiteren Versuchsreihe wurden Artikel aus dem wissenschaftlichen Journal *IEEE Transactions on „Medical Imaging“* herangezogen. Ein Beispiel einer Titelseite ist in Abbildung 7.3 dargestellt. Die logische und physische Struktur ist deutlich komplexer als im vorangegangenen Beispiel, denn die Artikel

enthalten Kapitel, Unterkapitel, Aufzählungen, mathematische Formeln und Literaturhinweise. Die physische Struktur wird für die Erkennung wiederum als Sequenz von physischen Entitäten repräsentiert, die den Textzeilen entsprechen. Allerdings sind die physischen Entitäten nun feiner klassifiziert, so dass sie neben dem Zeichensatz auch die horizontale Ausrichtung einer Zeile, bezogen auf die Textspalte, mit einbeziehen. Beispielsweise kodiert die physische Entität T11l1A eine Zeile, die vorwiegend die Schrift Times 11pt italic aufweist, links einfach eingerückt und rechts bündig mit dem Blockrand ist (siehe Tabelle 7.2). Von den insgesamt 192 möglichen physischen Entitäten werden in den „Medical Imaging“ Dokumenten jedoch nur 18 tatsächlich benötigt.

IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 14, NO. 3, SEPTEMBER 1995 471

An Improved Method for MRI Artifact Correction Due to Translational Motion in the Imaging Plane

Reza Aghaeizadeh Zoroofi, Yoshinobu Sato, *Member, IEEE*, Shinichi Tamura, *Member, IEEE*, Hiroaki Naito, and Li Tang

***Abstract*— A computer postprocessing technique is developed to remove MRI artifact arising from unknown translational motion in the imaging plane. Based on previous artifact correction methods, the improved technique uses two successive steps to reduce read-out and phase-encoding direction artifacts: First, the spectrum shift method is applied to remove read-out axis translational motion. Then, the phase retrieval method is employed to eliminate the remaining subpixel motion of the read-out axis and the entire motion of the phase-encoding axis. In the presence of noise, to protect edge detection (in the spectrum shift method), two high-density gray-level markers are added, one to each side of the imaging object. Experimental results with an actual MR scan confirmed the ability of the method to correct the artifact of an MR image caused by unknown translational motion in the imaging plane.**

I. Introduction

MAGNETIC resonance imaging (MRI) is widely used in medical diagnosis for its various advantageous features, such as high-resolution capability, the ability to produce an arbitrary anatomic cross-sectional image, and high tissue contrast [1]. Unfortunately, patient motion during MRI data acquisition (which takes several minutes) can limit the resultant image quality by imposing ghost-like artifact, blurring, or reducing the intensity of moving structures [2]. In some situations, unwanted motion of the imaging object during the MR scan is unavoidable. Undesired motion can be rigid and/or nonrigid. The unintentional translational or rotational motion of the head (of an infant or an injured person undergoing emergency treatment, for example) during MRI data acquisition can be considered as unwanted rigid motion, while the physiological movements of internal body organs (such as the heart and lungs) represent typical nonrigid motion. Generally, because patient motion is an unknown function of three-dimensional (3-D) space and time it is a very complex phenomenon to evaluate. For this reason, previous works have considered some of the simpler types of motional artifact. A good review of the methods developed to suppress motional artifact is given in [3]. With respect to two-dimensional (2-D) Fourier transform (2-DFT) MRI [4], [5], several recent papers have reported on the use of computer postprocessing to remove motional artifact [6]–[13]: Korin *et al.* [6] correct the phase of data using *a priori* knowledge about the motion; Hedley *et al.* assume that the motion occurs only along the slice selection axis [7] or in the imaging plane [8]–[10]; Mitsa *et al.* [11] propose a method to remove periodic motion artifact; Steagall *et al.* [12] use a projection onto a convex set (POCS) procedure in a constrained model of motion; and Tang *et al.* [13] deal with 2-D translational motion artifact correction.

Our goal is to reduce MRI artifact caused by the rigid motion of the object in the imaging plane. The assumed limitations on the motion are (similar to those given in [8]): 1) it is rigid, 2) it is translational, 3) it is in the imaging plane, and 4) it is an interview effect. The distinctive features of the method are: 1) no *a priori* knowledge about the motion is necessary, and 2) no modification is made to a standard pulse sequence.

Applying the above-mentioned assumptions, we have attempted to improve on existing computer postprocessing methods designed to correct MRI artifact due to translational motion in the imaging plane. First, we evaluate two previous methods and consider their capabilities and limitations in removing MRI translational motion artifact. We explain why the method of Hedley *et al.* [8] does not work well when the motion contains large variation (several pixels) along the read-out (X-direction) and phase-encoding (Y-direction) axes. We also discuss the limitations of the procedures proposed by Felmlee *et al.* [16] and Tang *et al.* [13] to correct subpixel motion along the read-out axis and substantial motion along the phase-encoding axis. We then combine the strong points of the above-mentioned procedures to introduce our improved method. The performance of the improved method is more satisfactory than the two previous methods in reducing MRI translational motion artifact in the imaging plane.

The paper is organized as follows: In Section II, we model the the problem of MRI 2-D translational artifact. In Section III, we review two previous methods and discuss their strong and weak points. Following a discussion in Section IV, in Section V we introduce an improved technique for dealing with the problem of MRI artifact due to rigid translational motion in the imaging plane. In Section VI, we verify our proposed method by experiments using both simulated and real MR images. The paper ends with concluding remarks.

II. Model of the Problem

The relation between the MRI signal and the density distribution of the target in the imaging plane is given by [6]

$$f(k_x, k_y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} p(x, y) \exp[-j2\pi(k_x x + k_y y)]\, dx\, dy, \quad (1)$$

Manuscript received July 20, 1994; revised June 9, 1995. The Associate Editor responsible for coordinating the review of this paper and recommending its publication was D. Nishimura.

The authors are with the Division of Functional Diagnostic Imaging, Biomedical Research Center, Osaka University Medical School, Suita, Osaka 565 Japan; e-mail: reza@image.med.osaka-u.ac.jp.

IEEE Log Number 9413894.

0278-0062/95$04.00 © 1995 IEEE

Abbildung 7.3: Titelseite eines eines typischen „Medical Imaging“ Dokuments.

Fontfamilie (Math, Times)		Fontgrösse		Fontattribute (normal, bold, italic, small caps)		linke Ausrichtung (adjusted, single/double indented)		rechte Ausrichtung (adjusted, not adjusted)
T M	×	10 11 12 20	×	N B I S	×	A 1 2	×	A N

Tabelle 7.2: Kodierung der Attribute von physischen Entitäten.

Der zu grossen Komplexität des Erkennungsalgorithmus aus der „Memento“ Versuchsreihe wurde damit begegnet, dass bei jedem Konstruktionsschritt eine komplette Sequenz von Knoten zu einem Unterbaum zusammengefasst wird (in ähnlicher Weise wie in Kapitel 6 beschrieben). Mit dieser Verbesserung ist es möglich, die um einiges längeren Dokumente in einer vertretbaren Zeit zu erkennen.

Insgesamt wurden sieben Artikel mit insgesamt 41 Seiten ausgewählt, deren physische Struktur, also die Sequenz von physischen Entitäten, manuell extrahiert wurde. Die Resultate einiger Durchgänge für die Erkennung der Makrostruktur sind in Tabelle 7.3 aufgelistet. Nach dem Lernen von zwei Dokumenten kann die Struktur von zwei weiteren Dokumenten mit einer Rate von 1% inkorrekter logischer Knoten erkannt werden. Die restlichen Dokumente sind nicht erkennbar, weil sie logische oder physische Knoten enthalten, die noch nicht gelernt wurden. Erst nach Einbezug eines weiteren Dokuments in das Modell sind alle Dokumente mit einer Fehlerrate von 0.7% erkennbar. Mit vier gelernten Dokumenten reduziert sich die Fehlerrate gar auf 0.4%.

Aus den Resultaten lässt sich schliessen, dass der modifizierte Algorithmus mit der direkten Gruppierung von Knoten die Komplexität reduziert. Dokumente mit bis zu tausend physischen Entitäten (Zeilen) sind ohne weiteres verarbeitbar. Die Modifikation führt aber auch dazu, dass die Struktur nicht mehr durchwegs korrekt erkannt werden kann, so dass etwa 1% der logischen Kno-

gelernte		erkannte		fehlerhafte Knoten	
Dokumente	Knoten	Dokumente	Knoten	absolut	relativ
2	376	2	1029	10	1%
3	461	4	1357	10	0.7%
4	629	3	1200	5	0.4%

Tabelle 7.3: Resultate der „Medical Imaging“ Testserie.

ten falsch konstruiert ist. Das Problem ist dem Algorithmus zuzuschreiben und nicht dem Modell, da die korrekten Lösungen gemäss der Generalized-Means-Funktion höhere Konformitätswerte aufweisen als die vom Algorithmus gelieferten.

Aus den aufgeführten Vorversuchen mit „Memento“ und „Medical Imaging“ Dokumenten lassen sich wertvolle Rückschlüsse auf das Verhalten und die Eigenschaften des Modells und des Erkennungsalgorithmus ziehen. Mit den gewonnenen Erfahrungen wurde ein Ansatz entwickelt, wie er in den Kapiteln 4–6 präsentiert wurde. Die folgenden Abschnitte behandeln nun die abschliessenden Tests.

7.2 Testdaten

Für die Durchführung der Haupttestreihe mussten Dokumente gesucht werden, die den folgenden Anforderungen genügen:

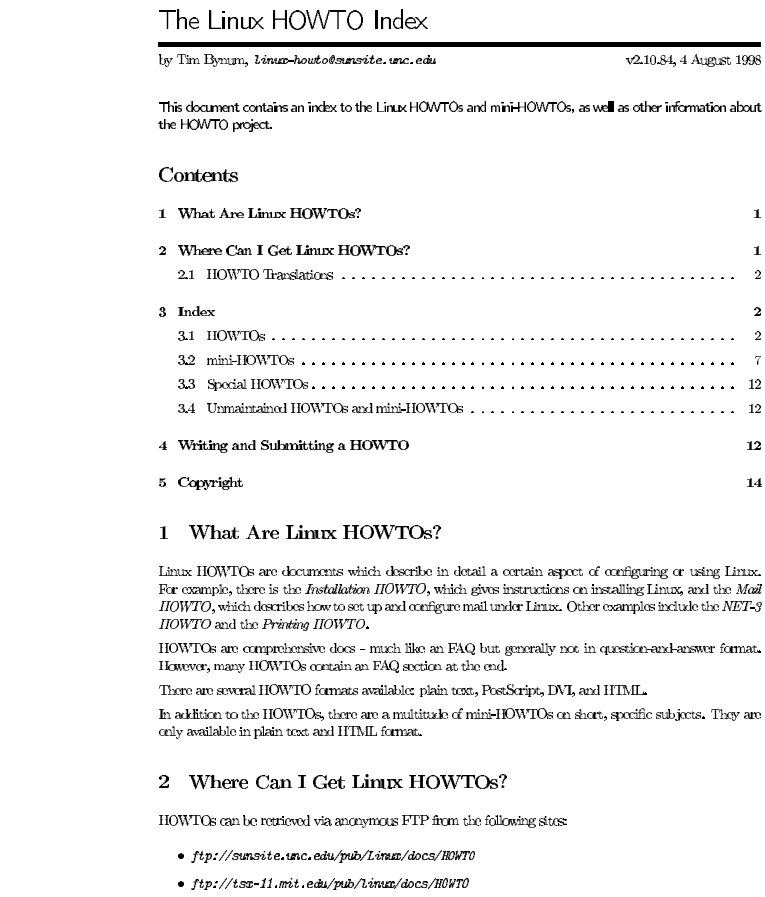
- Es muss eine grössere Anzahl, das heisst einige Dutzend, Dokumente verfügbar sein. Ein Dokument soll ungefähr ein Dutzend Seiten enthalten.
- Die Dokumente sollen eine reichhaltige, nicht triviale Struktur besitzen.
- Die Dokumente sollen in elektronischer Form vorliegen, damit auch eine Erkennung von idealen Dokumentbildern möglich ist. Das PostScript-Format soll vorhanden oder generierbar sein.
- Die Dokumente müssen alle derselben Klasse angehören und das gleiche Layout aufweisen, müssen also die gleiche logische und physische generische Struktur besitzen.
- Die generischen Strukturen müssen verfügbar sein, zum Beispiel in Form von grammatikalischen Regeln.

- Die Dokumente dürfen nicht künstlich generiert, sondern müssen von Autoren geschrieben sein.
- Um die Tests realistisch zu gestalten, sollten die Dokumente von verschiedenen Autoren verfasst worden sein.
- Die Dokumente sollen frei von zu restriktiven Copyrights sein.

Nach längerer Durchforstung der auf dem Internet frei erhältlichen Dokumente fiel unsere Wahl auf die Linux-HowTo Sammlung des LinuxDoc Projektes [Lin99]. Es handelt sich dabei um Anleitungen, die jeweils einen Aspekt des ebenfalls frei erhältlichen Betriebssystems Linux detailliert dokumentieren. Die Sammlung weist die folgenden, für unsere Zwecke wichtigen Vorteile auf:

- Die Dokumente sind in einer Vielzahl von Formaten verfügbar. Die Palette reicht von SGML, ASCII über HTML bis zu PostScript. Das allen gemeinsame Stammformat ist SGML.
- Die generische logische Struktur ist in Form einer Document Type Definition (DTD) verfügbar, die von Thomas Gordon aus der QWERTZ DTD [Her92] abgeleitet wurde.
- Die Programme, die die SGML-Dateien in andere Formate übersetzen, sind als Teil des SGMLTools-Paketes [SGM99] frei erhältlich, und können leicht auf jedem UNIX-System installiert werden.
- Die Sammlung umfasst zirka hundert Dokumente.
- Die Dokumente besitzen eine mittlere bis hohe Komplexität in ihrer Struktur.
- Um eine plattform- und medienunabhängige Präsentation zu gewährleisten, besitzen „LinuxDoc“ Dokumente keine Graphiken, Bilder und Tabellen. Dies ist für uns von Vorteil, da die Erkennung der genannten Elemente nicht Gegenstand unserer Forschungen ist, und sie ohnehin hätten ausgefiltert werden müssen.

Im Anhang A ist die DTD sowie einige Auszüge aus „LinuxDoc“ Dokumenten aufgeführt. Die Abbildung 7.4 zeigt eine typische Titelseite. Der in grosser Schrift geschriebene Titel (Title) ist gefolgt von einer Zeile mit Autor und Erstellungsdatum (Author). Danach folgt eine Zusammenfassung (Abstract) und das Inhaltsverzeichnis (TOC). Abstract und Inhaltsverzeichnis sind optional. Der Rest des Dokuments ist weiter unterteilt in Kapitel (Sect) und optional in Unterkapitel (Sect1) sowie Unter-Unterkapitel (Sect2), wie in den Abbildungen A.2



The Linux HOWTO Index

by Tim Bynum, *linux-howto@sunsite.unc.edu* v2.10.84, 4 August 1998

This document contains an index to the Linux HOWTOs and mini-HOWTOs, as well as other information about the HOWTO project.

Contents

1 What Are Linux HOWTOs?

Linux HOWTOs are documents which describe in detail a certain aspect of configuring or using Linux. For example, there is the *Installation HOWTO*, which gives instructions on installing Linux, and the *Mail HOWTO*, which describes how to set up and configure mail under Linux. Other examples include the *NET-3 HOWTO* and the *Printing HOWTO*.

HOWTOs are comprehensive docs - much like an FAQ but generally not in question-and-answer format. However, many HOWTOs contain an FAQ section at the end.

There are several HOWTO formats available: plain text, PostScript, DVI, and HTML.

In addition to the HOWTOs, there are a multitude of mini-HOWTOs on short, specific subjects. They are only available in plain text and HTML format.

2 Where Can I Get Linux HOWTOs?

HOWTOs can be retrieved via anonymous FTP from the following sites:

- *ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO*
- *ftp://tsx-11.mit.edu/pub/linux/docs/HOWTO*

Abbildung 7.4: Eine typische Titelseite mit Abstract und Inhaltsverzeichnis.

und A.3 ersichtlich ist. Die Textabsätze (P) sind mehrheitlich in der Schrift „Computer-Modern“ bei 10 Punkt Schriftgrösse gesetzt. Sie können aber zudem fast jede beliebige 10 Punkt Schrift enthalten (Computer-Modern fett und kursiv, sowie Courier und Script). Zwischen den Absätzen können vorformatierte Textteile (Verb) vorkommen, die ausschliesslich in Courier-Schrift gesetzt sind.

Des weiteren sind Beschreibungen (Descrip, Abbildung A.4), Auflistungen (Itemize, Abbildung A.5) und Aufzählungen (Enumerate, Abbildung A.6) zu finden. Sie bestehen aus einem eingerückten Absatz, dem ein Punkt bei Itemize,

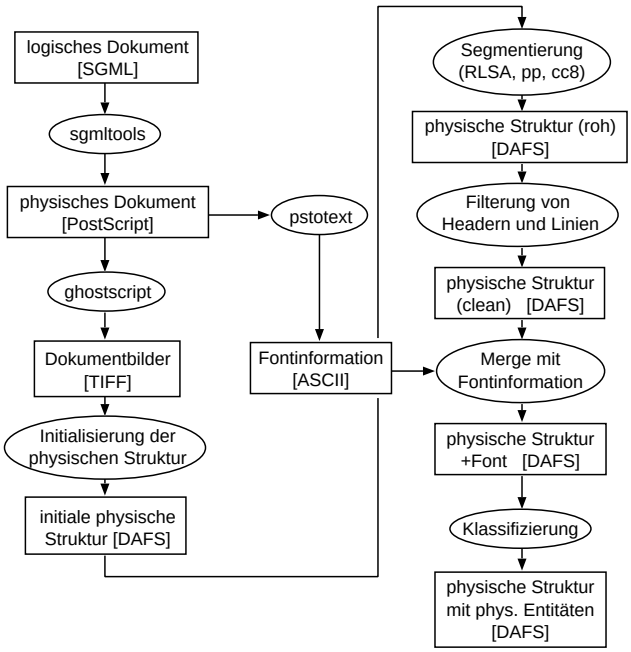


Abbildung 7.5: Blockschaltbild der Erkennung der physischen Struktur.

eine Zahl bei Enumerate oder ein kurzer, fett geschriebener Text (Tag) bei Descrip vorangestellt ist. Optional folgen ihnen weitere eingerückte Absätze (P) oder vorformatierte Textteile (Verb). Schliesslich enthalten einige Dokumente Zitate (Quote), die Absätzen gleichen, aber links und rechts eingerückt sind, wie das Beispiel in Abbildung A.7 zeigt.

7.3 Erkennung der physischen Struktur

Als Vorbedingung für die Erkennung der logischen Struktur eines Dokuments muss die physische Struktur vollständig bekannt sein. In diesem Abschnitt wird beschrieben, wie aus einem „LinuxDoc“ Dokument der in Kapitel 4 spezifizierte Strukturbaum generiert wird. Insbesondere wird die Klassifikation der Layoutattribute in physische Labels beschrieben.

Abbildung 7.5 stellt das Blockschaltbild der physischen Strukturerzeugung dar. Aus einem SGML-Dokument, das den Inhalt und die logische Struktur beschreibt wird mit den SGMLTools eine PostScript-Datei erzeugt. Intern ba-

ag-r-r-10	cm-r-r-10	cm-b-r-12	cr-r-r-8
ag-r-i-10	cm-r-i-10	cm-b-r-14	cr-r-r-10
ag-r-r-20	cm-b-r-10	cm-b-i-14	cr-r-r-12
	cm-b-i-10	bk-r-i-10	u1-r-r-10

Bedeutung der Abkürzungen		
f-w-a-s	f: family	ag: Avant-Garde, cm: Computer-Modern, cr: Courier, bk: Bookman script, u1: Special symbols
	w: weight	r: regular, b: bold
	a: attribute	r: roman, i: italic
	s: size	8, 10, 12, 14, 20 Punkt

Tabelle 7.4: Kodierung der Entitäten auf Wortniveau nach ihrem Font.

sieren die SGMLTools auf dem SGML-Parser *sp* [Cla99] und auf LaTeX als Layoutprozessor. Mit dem PostScript-Renderer *ghostscript* [Gho99] werden sodann Dokumentbilder in einer Auflösung von 300dpi erzeugt. Jedes Dokumentbild entspricht genau einer Seite. Die Bilder werden zu einer DAFS-Datei zusammengefasst. Das DAFS-Format eignet sich besonders für die Repräsentation von Dokumentstrukturen, da es gleichzeitig Bildinformationen und darauf überlagerte Baumstrukturen darzustellen vermag. Anschliessend wird jedes einzelne Bild in Zeilen, Wörter und Zeichen segmentiert. Wir verwenden dazu Routinen, die an unserem Institut implementiert wurden und auf den folgenden Standardmethoden basieren: Run-Length-Smoothing-Algorithmus (RLSA), Analyse von Projektionsprofilen (pp) und Extraktion 8-konnexer Komponenten (cc8). Die resultierenden Textblöcke entsprechen im „LinuxDoc“ Layout trivialerweise der konvexen Hülle aller Textzeilen, da die Dokumente immer einspaltig sind. Mit Hilfe von selbstentwickelten Filtern werden anschliessend die Kopfzeilen mit Kapitelüberschrift und Seitenzahl sowie horizontale Linien aus der Struktur entfernt. Schliesslich wird der DAFS-Struktur die Fontinformation hinzugefügt, die mit Hilfe des Tools *pstotext* [BML95] wortweise direkt aus der PostScript-Datei extrahiert wurde.

Zu diesem Zeitpunkt liegt die physische Struktur als Baum mit einer Tiefe von sechs vor. Auf dem ersten Niveau befindet sich die Dokumentwurzel, auf dem zweiten die Seitenknoten, auf dem dritten die Blockknoten (die hier exakt den Seiten entsprechen), auf dem vierten die Zeilenknoten, auf dem fünften

Abstand zur vorherigen Zeile n: normal, l: long, e:extended		Abstand zur folgenden Zeile n: normal, l: long, e:extended		linke Ausrichtung 0, 1, 2 oder 3-fache Einrückung		rechte Ausrichtung n: ausgerichtet, i: eingerückt
n e l	×	n e l	×	0 1 2 3	×	n i

Tabelle 7.5: Kodierung der Entitäten auf Zeilenniveau nach der relativen Position.

die Wortknoten und auf dem sechsten die Zeichenknoten, die den 8-konnexen Komponenten im Bild entsprechen. Alle Seiten-, Block, Zeilen-, Wort- und Zeichenknoten besitzen als Attribut die absolute Position des sie umschreibenden Rechtecks. Zusätzlich besitzen die Wortknoten als Attribut die Fontinformation sowie die Zeilenknoten die Position der Baseline des Textes.

Gemäss der Vorgaben für die Modellierung mit n-Grammen müssen die Attribute jedes physischen Knoten klassifiziert werden, woraus sich physische Entitäten ergeben. Die Knoten auf Dokument-, Seiten-, und Blockniveau werden nicht weiter klassifiziert und lediglich den Entitäten Volume, Page und Block zugewiesen. Die Knoten auf dem Wortniveau werden nach ihrer Fontinformation klassifiziert. Eine Klasse entspricht genau einem Font mit Attributen. Insgesamt existieren die 15 in Tabelle 7.4 aufgeführten Fontklassen.

Die Zeilenknoten werden schliesslich nach ihrer relativen horizontalen und vertikalen Position klassifiziert. Die Klassen sind, wie in Tabelle 7.5 gezeigt, als Zeichenkette der Länge vier dargestellt. Die erste Position widerspiegelt die vertikale Distanz einer Zeile zur vorangehenden Zeile und kann eine von drei Werten annehmen: n: normal, e: extended und l: long. Als Spezialfall wird einer ersten Zeile einer Seite, also direkt nach einem Zeilenumbruch der Wert n zugewiesen. Die zweite Position widerspiegelt in gleicher Weise die Distanz zur folgenden Zeile. Die dritte Position widerspiegelt den Abstand des linken Randes der Zeile von linken Rand des sie umgebenden Blockes. Sie kann vier Werte annehmen: 0: nicht eingerückt, 1: einfach eingerückt, 2: doppelt

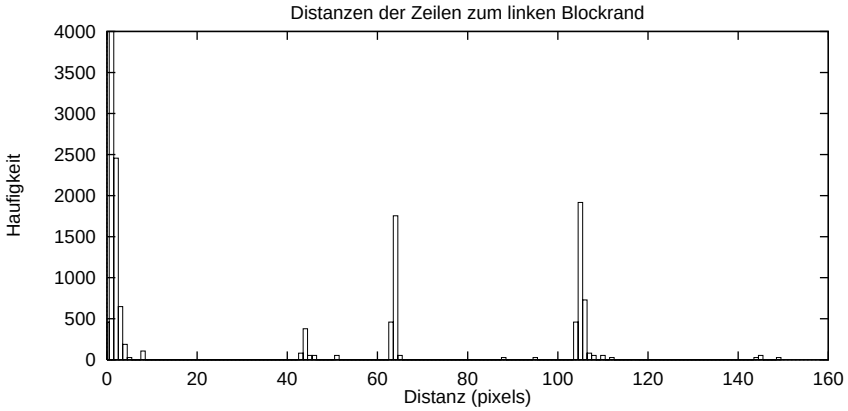


Abbildung 7.6: Verteilung der Abstandshäufigkeiten der linken Zeilenränder von den sie umgebenden Blöcken.

eingerückt und 3: dreifach eingerückt. Die vierte Position widerspiegelt den Abstand des rechten Randes der Zeile von rechten Rand des sie umgebenden Blockes. Sie kann zwei Werte annehmen: n: nicht eingerückt und i: eingerückt.

Die Klassifizierungen nach der relativen Position von Zeilen vollziehen sich durch einen einfachen Vergleich der gemessenen Distanzen mit Schwellwerten. Die Festlegung der Schwellwerte erfolgte in unseren Tests manuell, wurde jedoch von einer statistischen Analyse unterstützt. Im Diagramm in Abbildung 7.6 sind die Häufigkeiten der Abstände der linken Zeilenränder von ihren Blockrändern aufgezeichnet. Das Diagramm wurde mit zirka 20'000 Zeilen aus 27 Dokumenten berechnet. Es ist deutlich zu sehen, dass sich die Abstände in Gruppen einteilen lassen, die speziellen Formatierungen im Dokument entsprechen. Die Klassifizierung der Distanzen sollte so vorgenommen werden, dass die wichtigsten Gruppen in eigene Klassen fallen. Die Schwellenwerte für die Klassifizierung der linken Abstände in die Gruppen 0, 1, 2 und 3 wurden daher festgelegt auf 30 Pixel (differenziert zwischen der Klasse 0 für nicht eingerückt und 1 für einfach eingerückt), 80 Pixel (differenziert zwischen 1 und 2) und 140 Pixel (differenziert zwischen 2 und 3). Die Schwellenwerte für die Klassifikationen der vertikalen und rechten Zeilenabstände wurde analog vorgenommen.

7.4 Durchführung

Für die Realisierung der Tests wurde ein Prototyp erstellt, der im wesentlichen die in den Kapiteln 5 und 6 vorgestellten Algorithmen für das initiale und

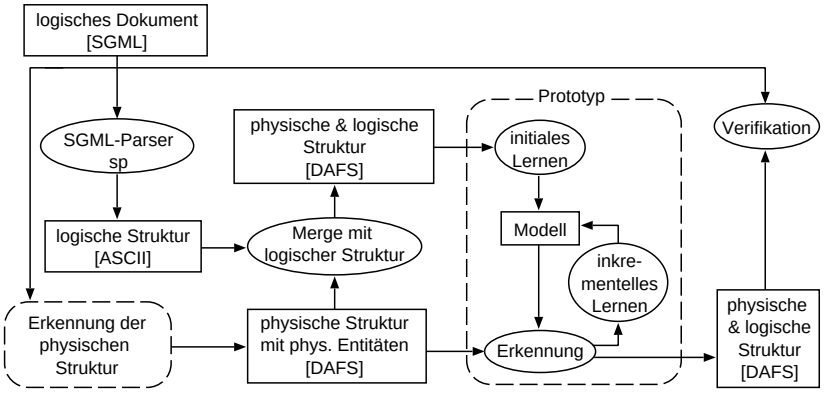


Abbildung 7.7: Blockschaltbild der Erkennung der logischen Struktur.

inkrementelle Lernen sowie für die Konstruktion der logischen Struktur implementiert. Wir betrachten nun den Ablauf eines Testdurchganges näher, der die Erkennung der Makrostruktur von „LinuxDoc“ Dokumenten zum Ziel hat.

Zu Beginn muss ein initiales Modell erstellt werden, das zumindest eine unvollständige Erkennung ermöglicht. Dazu wurden drei kleine Dokumente von fünf bis neun Seiten Umfang ausgewählt, die ein breites Spektrum von logischen Strukturelementen enthalten. Durch Parsing der SGML-Quelldateien mit dem SGML-Parser *sp* werden die entsprechenden Parsingbäume der drei Dokumente erzeugt. Nach Entfernung aller Elemente der Mikrostruktur muss jeder logischen Entität im Parsingbaum die zu ihr gehörige erste und letzte physische Entität manuell zugewiesen werden. Solchermassen vorbereitete Dokumente, die die vollständige physische und logische Struktur sowie die Zusammenhänge ihrer Elemente aufweisen, kann der Prototyp direkt lernen und daraus ein initiales Modell erzeugen. Der geschilderte Vorgang ist in Abbildung 7.7 schematisch dargestellt.

Mit einem initialen Modell kann das System nun selbständig die logische Struktur von Dokumenten erkennen. Schrittweise wird sie konstruiert, wobei der Benutzer den Erkennungsvorgang verfolgt und kontrolliert. Wenn das System im Laufe der Erkennung an einem gewissen Punkt aufgrund eines unvollständigen Modells keine weiteren Konstruktionsschritte durchführen kann, leitet es eine inkrementelle Lernphase ein.

Inkrementelles Lernen bedeutet konkret, dass der Benutzer manuell eine Gruppierung von logischen oder physischen Entitäten vornehmen muss, die das System alleine nicht vollziehen kann. Er definiert also eine Sequenz von En-

Parameter	Wert	Parameter beeinflusst ...
n	3	maximale Länge der n-Gramme
α	1	Messung der Modellkonformität von (partiellen) Lösungen
s	1	
splitdepth	$[1, 2]$	Aufteilung in Teilprobleme

Tabelle 7.6: Parameterwerte der Testdurchgänge.

titäten sowie das logische Label des Vorgängerknotens der Sequenz. In einem graphischen und interaktiven System ist dies mit wenigen Mausklicks zu bewerkstelligen. Das System integriert das neu hinzugekommene lokale Muster in sein n-Gramm-Modell und fährt mit der Erkennung fort.

Für die Tests wurden 26 Dokumente verschiedener Grösse ausgewählt, die insgesamt 390 Seiten und 8785 logische Entitäten umfassen. Die 26 Dokumente wurden in zwei Gruppen unterteilt. Die erste Gruppe besteht aus 15 Dokumenten, die keine Aufzählungen, keine Zitate und keine Einträge (Item) enthalten, die aus mehreren eingerückten Absätzen bestehen. Die zweite Gruppe besteht aus den verbleibenden elf Dokumenten. Die Einteilung in zwei Gruppen wurde vorgenommen, um den Effekt des inkrementellen Lernens besser beobachten zu können.

Vor der Durchführung von Testdurchgängen sind die Werte der insgesamt vier Parameter n, α, s und *splitdepth* festzulegen. In Vorversuchen hat sich herausgestellt, dass die Parameter den Verlauf der Erkennung nur schwach beeinflussen. Daher wurden alle Tests mit denselben Parameterwerten durchgeführt, die in Tabelle 7.6 aufgelistet sind.

7.5 Resultate

7.5.1 Fehlerrate

Als Mass für die Leistungsfähigkeit einer Erkennungsmethode wird normalerweise die Erkennungs- beziehungsweise die Fehlerrate berechnet. Da jedoch in unserem System Unklarheiten sofort interaktiv bereinigt werden, kann die resultierende logische Struktur grundsätzlich keine Fehler mehr enthalten. Daher messen wir nicht die Fehlermenge, die am Ende einer Erkennung gezählt wird, sondern die Anzahl der inkrementellen Lernschritte, die jeweils benötigt werden, um ein Dokument zu erkennen.

Das Diagramm in Abbildung 7.8 zeigt den Verlauf der Anzahl notwendi-

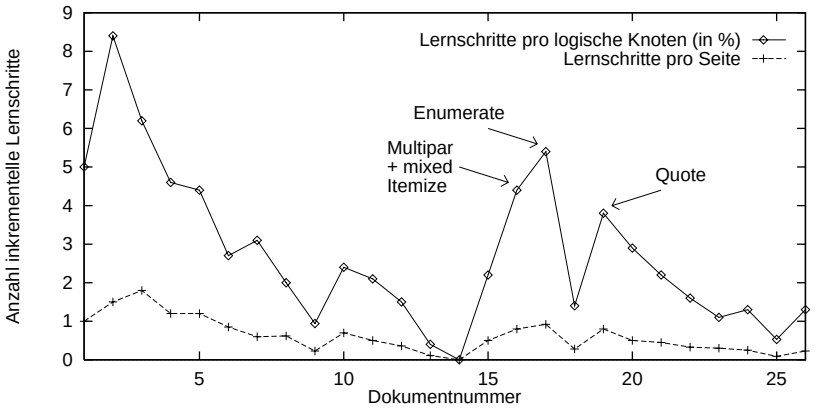


Abbildung 7.8: Entwicklung der Fehlerrate, beziehungsweise der Rate interaktiver Korrekturen mit zunehmender Anzahl erkannter Dokumente.

ger inkrementeller Lernschritte. Die durchgezogene Linie zeigt die Anzahl der Lernschritte bezogen auf die Anzahl logischer Entitäten im ganzen Dokument, und die gestrichelte Linie zeigt die Anzahl der Lernschritte pro Seite. Bei der Betrachtung des Verlaufs vom ersten bis zum fünfzehnten Dokument fällt auf, dass zu Beginn relativ viele Lernschritte benötigt werden, so dass zirka jeder zwanzigste logische Knoten von Hand definiert werden muss. Nach dem sechsten Dokument beginnt sich die Anzahl der Interaktionen erheblich zu reduzieren. Ab dem sechzehnten Dokument nimmt die Lernrate wieder deutlich zu. Die Erklärung dafür ist, dass neue, komplexe Strukturen in den Dokumenten erkannt werden müssen, die bis dahin noch in keinem Dokument auftraten. Es handelt sich um Item in einer Auflistung (Itemize), die aus mehreren Absätzen bestehen, die zusätzlich mit vorformatiertem Text (Verb) gemischt sind. Später kommen neu Aufzählungen (Enumerate) und Zitate (Quote) hinzu. Nach der Integration der neuen Strukturen nimmt die Rate der inkrementellen Lernschritte erneut ab und pendelt schliesslich um einen Wert von etwa 1%. Das heisst, eine manuelle Korrektur ist nötig bei jedem hundertsten logischen Knoten, was deutlich weniger als einer Korrektur pro Seite entspricht.

In einem letzten Durchgang wurde ein Modell aus den zuvor erkannten 26 Dokumenten gelernt. Dies ist möglich, da nach der Erkennung eines Dokuments die logische Struktur zusammen mit der physischen in einer DAFS-Datei gespeichert wird. In dieser Form kann sie direkt vom Prototypen geladen und gelernt werden. Mit dem Modell, das sich aus den 26 Dokumenten ergab, wurde erneut für jedes Dokument die logische Struktur konstruiert und mit der

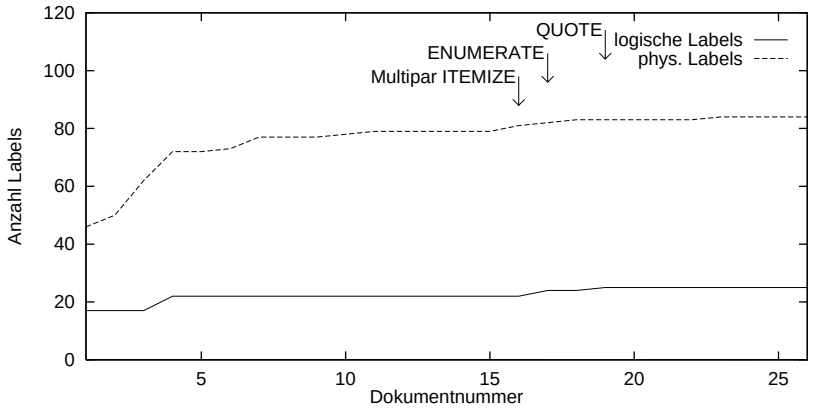


Abbildung 7.9: Die Anzahl logischer und physischer Klassen mit zunehmender Anzahl erkannter Dokumente.

originalen logischen Struktur verglichen. In 20 der 26 Fälle, also der überwiegenden Mehrzahl, wird exakt dieselbe logische Struktur konstruiert, wie sie im originalen Dokument vorlag. In drei Fällen wird gar eine korrektere Struktur konstruiert, da die originalen Dokumente, durch Nachlässigkeit des Benutzers in den inkrementellen Lernphasen, vereinzelt Fehler enthielten. In weiteren drei Fällen werden einige vorformatierte Textteile (Verb) in zwei oder mehr Verb-Entitäten aufgeteilt. Das Problem der Überfragmentierung von Verb-Entitäten wird weiter unten eingehend analysiert.

7.5.2 Modellkonvergenz

Die Diagramme in den Abbildungen 7.9 und 7.10 geben Aufschluss darüber, wie stark das Modell in den Lernphasen anwächst. Das erste Diagramm illustriert den Verlauf der Anzahl logischer und physischer Klassen. Die Anzahl der logischen Klassen entspricht direkt den SGML-Tags der Makrostruktur. Die Anzahl der physischen Klassen ist hingegen nicht direkt vorhersagbar. Im Diagramm wächst sie zunächst stark an, um dann zum Wert 84 zu konvergieren. Der theoretische Maximalwert beträgt 99 (81 Zeilenklassen, 15 Wortklassen und je eine für Block-, Seiten und Volumeentitäten). Der theoretische Maximalwert wird nicht erreicht, weil nicht alle Zeilenklassen in den Dokumenten auftreten.

Das Diagramm in Abbildung 7.10 zeigt den Verlauf der Anzahl der n-Gramme, die Wahrscheinlichkeiten grösser als 0 aufweisen. Die Kurven steigen zunächst stark an und gehen später in ein leichtes Wachstum über. Die theoreti-

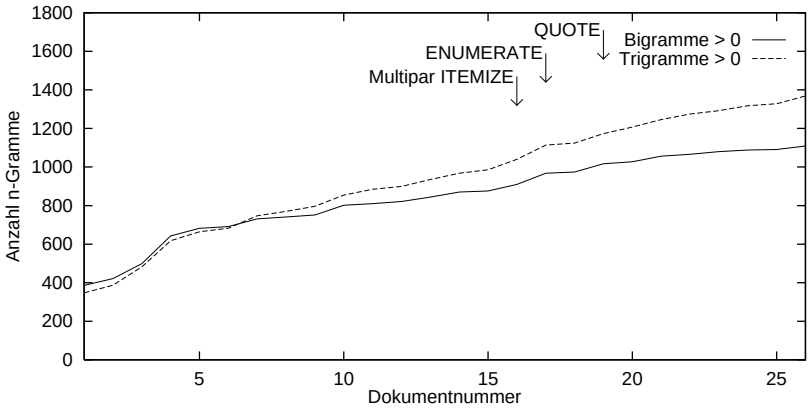


Abbildung 7.10: Die Anzahl der von 0 verschiedenen Bi- und Trigrammen mit zunehmender Anzahl erkannter Dokumente.

schen Maximalwerte betragen 11'425 für Bigramme und 767'025 für Trigramme. Das heisst, dass 9.7% aller möglichen Bigramme und 0.18% aller möglichen Trigramme einen von 0 verschiedenen Wert haben.

7.5.3 Entropie

In Tabelle 7.7 sind die Entropiewerte des Modells nach dem Lernen der 26 Dokumente aus den beiden Testgruppen aufgeführt. Die Tabelle führt sowohl die Gesamtentropie des Modells als auch die Entropien der n-Gramme für jede Nachbarschaftsbeziehung auf. Aus den Werten lassen sich einige interessante Schlüsse ziehen.

- Die Entropie von $r_{\lambda\mu\nu}$ ist leicht höher als die von $l_{\lambda\mu\nu}$. Der Grund liegt in den n-Grammen von Titelentitäten (Titel des Artikels, der Kapitel usw.), die nie einen linken Nachbarn besitzen. Die Entropie von $l_{\lambda\mu\nu}$ dieser Entitäten beträgt immer 0, was die Gesamtentropie stark reduziert. Werden die Titelentitäten nicht in Betracht gezogen, so ergibt sich für $r_{\lambda\mu\nu}$ eine höhere Entropie als für $l_{\lambda\mu\nu}$. Daraus lässt sich der Schluss ziehen, dass es sicherer ist, eine logische Entität vorauszusagen, wenn deren linker Kontext bekannt ist, als umgekehrt. Der Grund dafür liegt möglicherweise darin, dass Dokumentstrukturen derart gestaltet wurden, dass sie in der Lesefolge optimal verarbeitet werden können.

- Die Trigramme h^f und h^l besitzen eine niedrigere Entropie als i^f und

	Entropie	
	Bigramm	Trigramm
rechte Nachbarschaft, $r_{\lambda\mu\nu}$	0.57	0.56
linke Nachbarschaft, $l_{\lambda\mu\nu}$	0.49	0.46
Vorgänger, $t_{\lambda\mu\nu}$	0.21	0.12
erste Nachfolger, $b_{\lambda\mu\nu}$	0.0	0.0
letzte Nachfolger, $e_{\lambda\mu\nu}$	0.18	0.18
erste physische Entitäten, $h^f_{\mu\nu\vartheta}$	0.56	0.055
letzte physische Entitäten, $h^l_{\mu\nu\vartheta}$	0.94	0.24
sind erste physische Entitäten von, $i^f_{\varphi\vartheta\nu}$	0.42	0.10
sind letzte physische Entitäten von, $i^l_{\varphi\vartheta\nu}$	0.65	0.43
gesamtes Modell	0.47	0.26

Tabelle 7.7: Entropiewerte der Bi- und Trigramme eines Modells für eine Dokumentklasse mittlerer Komplexität.

i^l. Das bedeutet, dass sich mit dem Modell leichter physische Entitäten vorhersagen lassen, wenn der logische Kontext bekannt ist, als umgekehrt.

- Innerhalb der logischen Struktur führt die Erhöhung von n von 2 auf 3 nur zu einem marginal aussagekräftigeren Modell. Dies ist insofern erstaunlich, als man davon ausgehen würde, dass die Berücksichtigung eines tieferen Kontexts zu präziseren Vorhersagen führte. Offensichtlich sind die Beziehungen in der logischen Struktur sehr lokal. Hingegen kann das Modell für die Beziehung zwischen logischer und physischer Struktur durchaus von einer Erhöhung von n profitieren.

- Die Entropie von b für die Beschreibung des ersten Nachfolgerknotens beträgt 0, weil alle n-Gramme 0 oder 1 betragen und somit die Struktur perfekt beschreiben.

- Die ersten Entitäten von Sequenzen werden vom Modell schärfer beschrieben, als die letzten Entitäten, da $H(b) < H(e)$, $H(h^f) < H(h^l)$ und $H(i^f) < H(i^l)$.

- Generell werden vertikale Muster vom Modell besser erfasst als horizontale, da die Entropie von r und l generell grösser ist als die von t, b, e, h^f,

h^l, i^f und i^l.

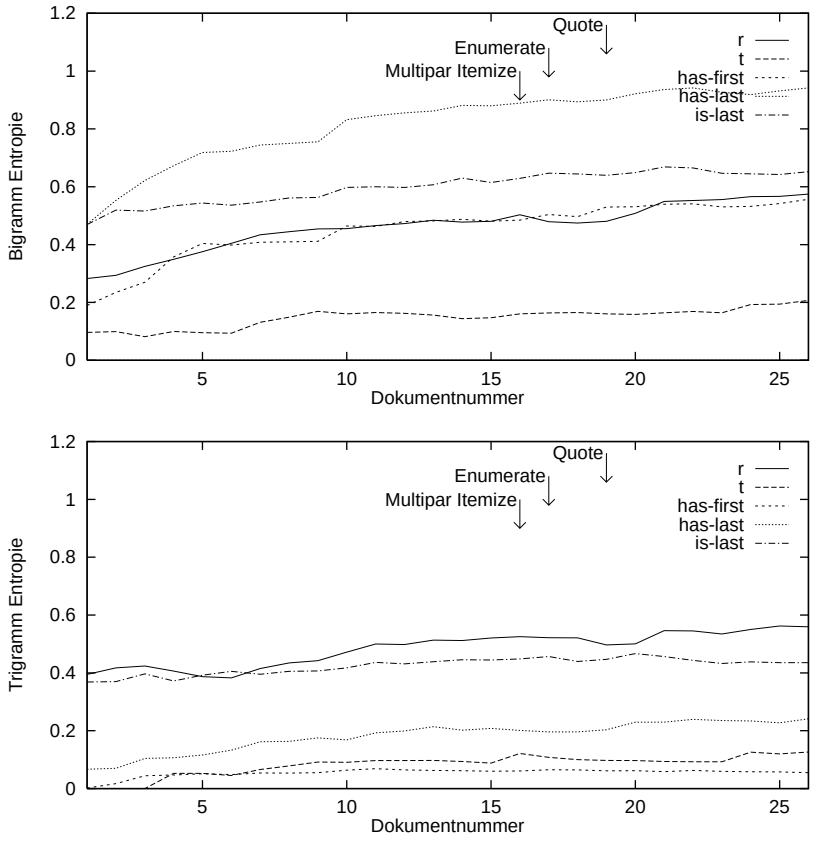


Abbildung 7.11: Die Entwicklung der Entropie von Bi- und Trigrammen mit zunehmender Anzahl erkannter Dokumente.

Bei der Betrachtung des zeitlichen Verlaufs einiger Entropiewerte in den Diagrammen der Abbildung 7.11 fällt zunächst auf, dass teilweise erhebliche Schwankungen auftreten. Insgesamt steigen die Kurven mit dem Lernen der ersten Dokumente stark an. Nach etwa einem Dutzend gelernter Dokumente beginnen die Entropiewerte allmählich zu konvergieren. Die n-Gramm-Wahrscheinlichkeiten scheinen also mit fortschreitendem Hinzulernen von lokalen Mustern ihre Trennschärfe zu erhalten und nicht zu „verwischen“. Dies ist ein zusätzlicher Hinweis dafür, dass das Modell mit der Zeit konvergiert.

7.5.4 Effizienz

Ein wichtiger Kennwert für den Erkennungsalgorithmus ist dessen praktische Effizienz, insbesondere deshalb, weil die theoretische Komplexität exponentiell mit der Grösse des Problems anwächst, und da ein typisches 15-seitiges Dokument etwa 600 physische Entitäten enthält. Da der Algorithmus jedoch äusserst restriktiv Hypothesen für die Gruppierung und Generierung von Knoten erzeugt, bleibt die praktisch gemessene Komplexität sehr niedrig. Im Durchschnitt werden nur doppelt so viele Konstruktionsschritte durchgeführt, wie die Lösung logische Knoten enthält. Als Maximalwert werden bei einem Dokument 2.8 Mal so viele Konstruktionsschritte wie logische Knoten gemessen. Zudem korreliert der genannte Faktor nicht mit der Grösse des zu erkennenden Dokuments. In absoluten Zeiten ausgedrückt heisst dies, dass beispielsweise ein längeres, 30-seitiges Dokument in wenigen Sekunden komplett erkannt wird; vorausgesetzt, das Modell ist vollständig.

Andererseits hat der Algorithmus einen vergleichsweise hohen Speicherbedarf. Eine partielle Lösung, die einem Knoten im Suchbaum entspricht, kann in grossen Dokumenten ohne weiteres mehrere hundert kByte belegen. Wenn der Suchbaum in einem extremen Fall mehrere tausend Knoten enthält, so sind effiziente Speicherverwaltungstechniken nötig, um die Grenzen gängiger Arbeitsplatzrechner nicht zu sprengen. Der maximale Arbeitsspeicherbedarf betrug in unseren Versuchen maximal lediglich 60 MByte, und dies obwohl in der gegenwärtigen Implementierung ein noch grosses Optimierungspotential nicht ausgeschöpft wurde.

7.6 Probleme bei der Erkennung

In seltenen Fällen gelingt es dem System trotz inkrementeller Lernschritte nicht, die korrekte Lösung zu konstruieren. So werden vorformatierte Textteile (Verb) oft als mehrere Verb-Entitäten erkannt, wenn sie durch einen grösseren Abstand voneinander getrennt sind (Abbildung 7.12). Der Grund dafür ist, dass der Autor eines Dokuments des initialen Lernsets einige Male direkt aufeinanderfolgende Verb-Entitäten verwendet hat. Wenn eine solche Struktur einmal gelernt wurde, ist sie anschliessend durch inkrementelle Lernschritte nicht mehr aus dem Modell zu entfernen.

Ein weiterer, seltener Fehlertyp tritt in Textteilen auf, wie er in Abbildung 7.13 dargestellt ist. Die zwei Zeilen vor dem Titel müssten zu einem Absatz (P) zusammengefasst werden. Da das Modell aber nur die ersten und letzten physischen Entitäten einer logischen Entität repräsentiert, interpretiert das System die Zeilen als vorformatierten Text (Verb). Im vorliegenden Fall sollte es der Benutzer aber vermeiden, die beiden Zeilen als Absatz (P) lernen

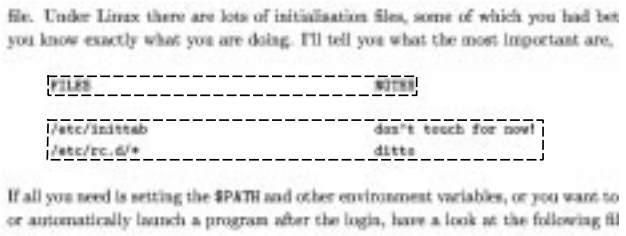


Abbildung 7.12: Eine in zwei Entitäten geteilte Verb-Entität.

zu lassen; sonst wird riskiert, dass danach Verb-Entitäten fälschlicherweise als P-Entitäten erkannt werden. Im Kapitel 9 über Erweiterungen wird vorgeschlagen, wie mit einem Ausbau des Modells dieses Problem behoben werden kann.

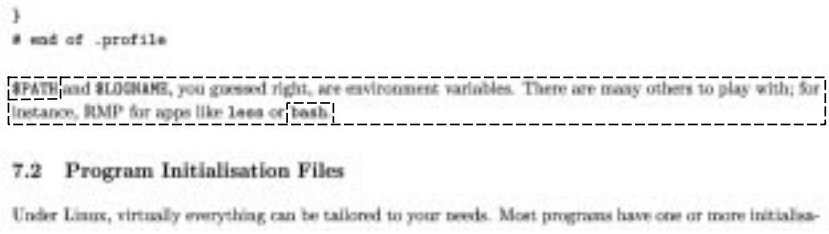


Abbildung 7.13: Eine als Verb erkannte P-Entität.

Die Fehlertoleranz des Algorithmus bleibt hinter den Erwartungen zurück. Dies hat sich in Fällen gezeigt, in denen durch einen Fehler in der Assoziierung des Font zu Wörtern die physische Struktur teilweise fehlerhaft aufgebaut wurde. Solche Dokumente können erst richtig erkannt werden, wenn vorgängig die Fontinformation korrigiert wird.

Eine gewisse Toleranz zeigt sich, wenn beispielsweise beim Zusammenfassen von Knoten zu einem Absatz darin unbekannte, das heisst ungelernte, Wort- oder Zeilenentitäten auftreten. Der Algorithmus integriert diese einfach in den Absatz, in dem sie sich befinden. Ebenso reagiert der Algorithmus auf fehlerhafte Wortsegmentierungen und teilweise auch auf fehlerhafte Zeilensegmentierungen sehr robust.

7.7 Zusammenfassung

Unser Ansatz zur Modellierung, das Lernen und die Erkennung von Dokumentstrukturen wurde an einer Reihe verschiedener Dokumentklassen evaluiert. Die Haupttests wurden an 26 Dokumenten mit insgesamt 390 Seiten der „LinuxDoc“ Sammlung durchgeführt. Die Dokumente weisen in ihrer Struktur eine mittlere Komplexität auf und enthalten neben den üblichen Strukturen auch Listen, Aufzählungen und Beschreibungen.

Die Resultate zeigen, dass das n-Gramm-Modell die Makrostruktur der Dokumente adäquat repräsentieren kann. Mit einem Modell, das mit drei kleinen Dokumenten initialisiert wird, ist es möglich, benutzerassistierte Erkennungssessionen durchzuführen. Muss der Benutzer bei der Erkennung der ersten drei bis fünf Dokumente mit zirka eineinhalb inkrementellen Lernschritten pro Seite noch relativ häufig intervenieren, so reduziert sich diese Rate nach zwanzig Dokumenten auf zirka einen inkrementellen Lernschritt alle vier Seiten. Aus dem Verlauf der Modellentropie und weiteren Modellkennwerten lässt sich schliessen, dass das Modell mit fortlaufender Anwendung in einen stabilen Zustand konvergiert. Trotz der theoretisch exponentiellen Komplexität des Algorithmus sind in der Praxis nur wenige Konstruktionsschritte nötig. Empirisch wird gar eine lineare Komplexität gemessen. Somit lässt sich mit einem vollständigen Modell ein gesamtes Dokument in wenigen Sekunden erkennen.

Kapitel 8

Integration mit Dokumentstandards

Im einführenden Kapitel 2 über Dokumentstrukturen und ihre Verarbeitung wurde dargelegt, dass SGML der zur Zeit etablierte Standard für die Spezifikation und Repräsentation von logischen Strukturen ist. Die meisten professionellen Textverarbeitungen können SGML-Dateien lesen und schreiben, und zusätzlich existiert eine unüberschaubar grosse Menge an Werkzeugen und Bibliotheken für die Verarbeitung von SGML-Dateien. Hinweise dazu sind in der *SGML/XML Web Page* von Robin Cover zu finden [Cov99], die das wohl vollständigste Archiv rund um SGML und XML darstellt.

Es ist daher naheliegend, die Integrationsmöglichkeiten des von uns vorgeschlagenen Ansatzes für die Dokumenterkennung zu erforschen. Die einfachste Form der Integration ist die Bereitstellung der erkannten Dokumente im SGML-Format. Dies ist trivial, da die SGML-Struktur exakt dem konstruierten logischen Strukturbaum entspricht, der sich damit ohne weiteres direkt im entsprechenden ASCII-Format ausgeben lässt. Nun kann aber das Potential eines SGML-Dokuments erst richtig ausgeschöpft werden, wenn auch die dazugehörige DTD verfügbar ist, da diese von fast allen SGML-verarbeitenden Programmen benötigt wird. Der anspruchsvolle Teil der Integration besteht also in der *Inferenz* einer DTD aus einem n-Gramm-Modell für Dokumentstrukturen. Die dazu an unserem Institut durchgeführten Experimente werden im Abschnitt 8.1 vorgestellt. Anschliessend wird im Abschnitt 8.2 die Möglichkeit der Inferenz eines typographischen Modells in Form einer Spezifikation in DSSSL diskutiert. Schliesslich wird in Abschnitt 8.3 besprochen, wie ein n-Gramm-Modell mit einer vorliegenden DTD und DSSSL-Spezifikation initialisiert werden könnte.

8.1 Inferenz von Document Type Definitions

Im Rahmen einer Diplomarbeit wurde von Daniel Bollinger ein Algorithmus für die Inferenz von DTDs entwickelt und implementiert [Bol96]. Zwei Hauptziele waren ihm gesetzt.

1. Der Inferenzalgorithmus soll seine Informationen möglichst nur aus dem n-Gramm-Modell beziehen.
2. Die erzeugte DTD soll sich von einer manuell erstellten DTD möglichst wenig unterscheiden.

Die anschliessenden Abschnitte erläutern das Grundprinzip des Algorithmus und die Ergebnisse, die mit „Medical Imaging“ und „LinuxDoc“ Dokumenten erzielt wurden. Schliesslich werden die noch offenen Hauptprobleme der Inferenz diskutiert.

8.1.1 Das Prinzip

Die meisten in der Literatur anzutreffenden Inferenzalgorithmen für DTDs basieren auf dem Prinzip, dass aus Lernsamples eine initiale, kontextfreie Grammatik erzeugt wird, die exakt nur diese Samples akzeptiert. Diese Grammatik wird anschliessend so generalisiert, dass sie die gesamte Klasse der gelernten Dokumentstrukturen akzeptiert [Aho96, Sha95]. In Bollingers Algorithmus wird jedoch in umgekehrter Weise verfahren. Er erzeugt aus den n-Gramm-Wahrscheinlichkeiten eine initiale Grammatik, die zu allgemein ist, und spezialisiert sie anschliessend durch gezielte Interpretation der n-Gramme.

Bei der Entwicklung des Algorithmus hat sich herausgestellt, dass sich die Inferenz durch eine minimale Erweiterung des Modells erheblich verfeinern lässt. Konkret wird zusätzlich zu den n-Grammen die minimale und maximale Anzahl von Nachfolgerknoten jeder logischen Entität im Strukturbaum benötigt.

Die initiale Grammatik besteht aus je einer Produktionsregel für jede logische Entität. Ihre allgemeine Form ist:

$$\begin{aligned} \lambda_i \quad &\rightarrow \quad (\lambda_{j_1} \mid \ldots \mid \lambda_{j_I})^*, \\ &\text{mit} \quad \lambda_{j_x} \in \{\lambda_j \mid t_{\lambda_j \lambda_i} > 0\}. \end{aligned}$$

Die initialen Regeln drücken aus, dass jede logische Entität λ_j in eine beliebig lange Sequenz aller seiner möglichen Nachfolgeentitäten umgeschrieben werden kann. Die möglichen Nachfolgeentitäten sind diejenigen Entitäten λ_j die λ_i als Vorgängerknoten besitzen können ($t_{\lambda_j \lambda_i} > 0$).

Um die Dokumentstruktur exakter zu beschreiben, werden diese sehr generellen Regeln anschliessend spezialisiert. In insgesamt sieben Spezialisierungsstufen werden die ursprünglichen Regeln unter Verwendung der Informationen im

n-Gramm-Modell schrittweise umgeformt. Eine Spezialisierung bedeutet, dass bei jeder Umformung die von den Regeln akzeptierte Sprache kleiner wird. Das heisst, die Sprache, die nach der Umformung akzeptiert wird, ist eine Teilmenge der zuvor akzeptierten. Die Spezialisierungen besitzen zudem die wichtige Eigenschaft, dass sie keine Heuristiken oder Schätzungen in den Umformungen anwenden, sondern exakte Ableitungen aus den bestehenden Produktionsregeln und dem n-Gramm-Modell vornehmen.

Ein Beispiel einer Umformung basiert auf der Interpretation der Bigramme b und e, die den Beginn und das Ende der Sequenz von Nachfolgeknoten einer Entität λ_i beschreiben. So muss das erste Symbol einer Produktion notwendigerweise eine Entität λ_k sein, die ein erster Nachfolger von λ_i sein kann ($b_{\lambda_i\lambda_k} > 0$). Das Analoge gilt für das letzte Symbol einer Produktion.

$$\begin{aligned} \lambda_i \quad &\rightarrow \quad ((\lambda_{k_1} \mid \ldots \mid \lambda_{k_K}), (\lambda_{j_1} \mid \ldots \mid \lambda_{j_J})^*, (\lambda_{m_1} \mid \ldots \mid \lambda_{m_M})), \\ &\text{mit} \quad \lambda_{k_x} \in \{\lambda_k \mid b_{\lambda_i\lambda_k} > 0\}, \quad \lambda_{m_x} \in \{\lambda_m \mid e_{\lambda_i\lambda_m} > 0\}. \end{aligned}$$

Für eine detaillierte Beschreibung der restlichen spezialisierenden Umformungen sei auf Bollingers Schlussbericht des Projekts verwiesen.

Wenn keine weiteren spezialisierenden Umformungen mehr möglich sind, werden die Regeln für die Ausgabe vorbereitet. Das heisst, dass eine Reihe von Substitutionen auf die Regeln angewendet werden, die zum Ziel haben, sie in eine gut leserliche Form zu transformieren. Die Substitutionen verwenden dafür keinerlei Information mehr aus dem n-Gramm-Modell. Beispiele für Substitutionen sind:

$$\begin{aligned} A \rightarrow BB^* \quad &\rightsquigarrow \quad A \rightarrow B^+ \\ A \rightarrow B^*B \quad &\rightsquigarrow \quad A \rightarrow B^+ \\ A \rightarrow B \mid BC^+ \quad &\rightsquigarrow \quad A \rightarrow BC^* \\ A \rightarrow B \mid C^+B \quad &\rightsquigarrow \quad A \rightarrow C^*B \\ A \rightarrow BCCC\ldots D \quad &\rightsquigarrow \quad A \rightarrow BC^+D. \end{aligned}$$

Alle obigen Substitutionen verändern die von den Regeln akzeptierte Sprache nicht, mit Ausnahme der letzten. Ihr liegt die generalisierende Heuristik zugrunde, nach der eine Sequenz von drei oder mehr gleichen Entitäten als Iteration zu interpretieren ist.

Schliesslich werden nach den abgeschlossenen Spezialisierungen und Substitutionen die generierten Regeln in Form einer DTD ausgegeben. Der folgende Abschnitt präsentiert nun einige Resultate der Regelinferenz.

8.1.2 Resultate

Die DTD-Inferenz wurde mit allen Dokumentklassen durchgeführt, die im Zusammenhang mit der Evaluation in Kapitel 7 besprochen wurden. Die Tabelle 8.1 zeigt eine Gegenüberstellung zweier DTDs für die Dokumentklasse „Medical Imaging“. Links ist eine manuell erstellte DTD abgebildet, und rechts die vom System automatisch erzeugte[1].

Von Hand erstellte DTD	Durch Inferenz erzeugte DTD
`Article -> Head, Body, Refs`	`Article -> Head, Body, Refs`
`Head -> Title, Authors, Abstract`	`Head -> Title, Authors, Abstract`
`Body -> Chap+`	`Body -> Chap, Chap+`
`Chap -> Chaptit, Parag*, Schap*`	`Chap -> Chaptit, (Parag | Schap)+`
`Schap -> Schaptit, Parag+`	`Schap -> Schaptit, Parag+`
`Refs -> Reftit, Refentry+`	`Refs -> Reftit, Refentry+`
`Title -> T2OBC+`	`Title -> T2OBC, T2OBC+`
`Authors -> T12NC+`	`Authors -> (T12NC, T12NC) | T12NC`
`Abstract -> T11B1A, T11BAA*, T11BAN`	`Abstract -> T11B1A, T11BAA*, T11BAN`
`Chaptit -> T11SC`	`Chaptit -> T11SC`
`Schaptit -> T11IAN`	`Schaptit -> T11IAN`
`Parag -> Par | Enum`	`Parag -> Par | Enum`
`Par -> (T11N1A, (T11NAA | T11NAN | M11NC)*) | T11N1N`	`Par -> T11N1A, (T11NAA | T11NAN | M11NC)* , T11NAN`
`Enum -> Enentry+`	`Enum -> Enentry, Enentry+`
`Enentry -> (T11N1A, (T11N2A | T11N2N| M11NC)*) T11N1N`	`Enentry -> (T11N1A, T11N2A*, T11N2N) | T11N1N`
`Reftit -> T11SC`	`Reftit -> T11SC`
`Refentry -> (T1ONAA, T1ON1A*, T1ON1N) | T1ONAN`	`Refentry -> (T1ONAA, T1ON1A*, T1ON1N) | T1ONAN`

Tabelle 8.1: Vergleich der DTDs für „Medical Imaging“ Artikel (aus Platzgründen nicht in SGML konformer Syntax dargestellt).

Zunächst fällt auf, dass mehr als die Hälfte der durch Inferenz erzeugten Regeln exakt den manuell erstellten entsprechen. Manche Regeln wurden hingegen nicht so stark generalisiert, wie das ein Autor einer DTD tun würde. So

[1] Wie im Kapitel über die Evaluation schon erwähnt wurde, bestand bei den „Medical Imaging“ Tests noch keine Unterscheidung zwischen einer physischen und logischen Ebene. Vielmehr wurden physische und logische Entitäten im gleichen Baum dargestellt, wobei die physischen Entitäten (wie beispielsweise `T2OBC`) die Blätterpositionen besetzen. Da der eigentliche Textinhalt in SGML als *parsed character data* (`#PCDATA`) deklariert wird, müssten Regeln wie `Title` → `T2OBC+` korrekt als `Title` → `#PCDATA` geschrieben werden. Die Regeln wurden allerdings in der ursprünglichen Form belassen, da sie ein limitiertes Layoutmodell definieren.

```
<!ELEMENT Article  -- (((Title, Author, Abstract)|(Title, Author)),
                       (Sect|Toc)*, Sect)>
<!ELEMENT Title    -- (#PCDATA)>
<!ELEMENT Author   -- (#PCDATA)>
<!ELEMENT Abstract -- (#PCDATA)>
<!ELEMENT Toc      -- (Toctit, (Tocsect|Tocsect1|Tocsect2)*)>
<!ELEMENT Toctit   -- (#PCDATA)>
<!ELEMENT Tocsect  -- (#PCDATA)>
<!ELEMENT Tocsect1 -- (#PCDATA)>
<!ELEMENT Tocsect2 -- (#PCDATA)>
<!ELEMENT Sect     -- (Secthd, (Enumerate|Descrip|Itemize|Verb|Sect1|P)+)>
<!ELEMENT Secthd   -- (#PCDATA)>
<!ELEMENT P        -- (#PCDATA)>
<!ELEMENT Verb     -- (#PCDATA)>
<!ELEMENT Sect1    -- (Sect1hd,
                       (P|Quote|Enumerate|Itemize|Sect2|Verb)*,
                       (P|Quote|Enumerate|Descrip|Itemize|Sect2|Verb)
                      )>
<!ELEMENT Sect1hd  -- (#PCDATA)>
<!ELEMENT Itemize  -- (Item, (Dp|Item|Verb)*)>
<!ELEMENT Item     -- (#PCDATA)>
<!ELEMENT Sect2    -- (Sect2hd,
                       (P|Itemize|Verb)*,
                       (P|Itemize|Verb|Enumerate)
                       )>
<!ELEMENT Sect2hd  -- (#PCDATA)>
<!ELEMENT Dp       -- (#PCDATA)>
<!ELEMENT Enumerate-- (Enum, (Enum|Dp|Verb)+)>
<!ELEMENT Enum     -- (#PCDATA)>
<!ELEMENT Descrip  -- (Tag,  (Dp|Tag|Verb)*, Dp)>
<!ELEMENT Tag      -- (#PCDATA)>
```

Tabelle 8.2: Durch Inferenz erzeugte DTD für „LinuxDoc“ Dokumente.

deklariert das System, dass der Textkörper mindestens zwei Kapitel enthalten muss ($\mathsf{Body} \rightarrow \mathsf{Chap}, \mathsf{Chap}^+$), während üblicherweise in DTDs auch ein einzelnes Kapitel erlaubt wird ($\mathsf{Body} \rightarrow \mathsf{Chap}^+$). Da jedoch nie ein Dokument gelernt wurde, das nur ein Kapitel enthält, generiert das System eine leicht restriktivere Regel. Andererseits sind auch Fälle auszumachen, in denen das System eine zu permissive Regel generiert. So erlaubt es in seiner DTD, dass in einem Kapitel auf den Titel eine Iteration von Absätzen und Unterkapiteln folgt ($\mathsf{Chap} \rightarrow \mathsf{Chaptit}\ (\mathsf{Parag} \mid \mathsf{Schap})^+$). Typische Dokumentstrukturen würden hingegen keine Absätze auf der gleichen Strukturebene nach einem Unterkapitel erlauben ($\mathsf{Chap} \rightarrow \mathsf{Chaptit}\ \mathsf{Parag}^*\mathsf{Schap}^*$).

Bei der Inferenz der DTD für die „LinuxDoc“ Dokumente sind deutliche

Unterschiede zwischen der originalen, manuell erstellten DTD (siehe Anhang A) und der automatisch erzeugten in Tabelle 8.2 zu beobachten. Grundsätzlich ist festzuhalten, dass die erzeugten Regeln meist zu spezifisch sind. So kann beispielsweise gemäss erzeugter DTD:

- ein Zitat (Quote) nur in einem Unterkapitel (Sect1) vorkommen,
- keine Beschreibung (Descrip) direkt auf einen Titel eines Unterkapitels (Sect1hd) folgen und
- keine Beschreibung (Descrip) mit vorformatiertem Text (Verb) enden.

Die Gründe dafür sind einerseits in den Spezialisierungsschritten der Inferenz zu suchen, die zuweilen offensichtlich über das Ziel hinausschiessen. Andererseits würden diese Fehler nicht auftreten, wenn das n-Gramm-Modell vollständiger wäre. So ist es eine Frage der Zeit, bis das System beim Lernen oder Erkennen auf ein Dokument stösst, das auch in einem Kapitel (Sect) ein Zitat (Quote) enthält.

8.1.3 Mehrdeutigkeit einer DTD

Wenn in einer kontextfreien Grammatik ein Wort existiert, dass zwei verschiedene Parsingbäume besitzt, so ist die Grammatik *mehrdeutig*. Der SGML-Standard schreibt vor, dass das Modell in einem strengeren Sinne nicht mehrdeutig sein darf. Es wird verlangt, dass ein Element in einer Dokumentinstanz eindeutig einem mehrfach auftretenden Element im Modell zugewiesen werden kann und zwar ohne look-ahead, das heisst, ohne in der Dokumentinstanz vorauszuschauen um zu entscheiden, welchem Element im Modell es zugewiesen werden soll (siehe Beispiel weiter unten). Eine SGML-Grammatik gehört folglich zur Klasse der kontextfreien LL(1) Grammatiken [ASU86].

Eine durch Inferenz aus dem n-Gramm-Modell erzeugte DTD ist im allgemeinen mehrdeutig. In der folgenden Regel aus Tabelle 8.2

```
<!ELEMENT Sect2     -- (Sect2hd,
                        (P|Itemize|Verb)*,
                        (P|Itemize|Verb|Enumerate)
                       )>
```

kann beispielsweise für einen Absatz (P), der gleich auf einen Titel eines Unter-Unterkapitels (Sect2hd) folgt, nicht entschieden werden, ob er dem Term `(P|Itemize|Verb)*` oder `(P|Itemize|Verb|Enumerate)` zugewiesen werden soll. Die Regel

```
<!ELEMENT Sect2    -- (Sect2hd,
                       (P|Itemize|Verb)+, Enumerate?
                      )>
```

ist mit der vorherigen insofern identisch, als sie exakt dieselbe Sprache akzeptiert. Sie ist jedoch eindeutig.

Insgesamt ist eine der 17 Regeln der „Medical Imaging“ DTD und vier der 24 Regeln der „LinuxDoc“ DTD mehrdeutig. Für die Aufhebung der Mehrdeutigkeiten stehen zwei Möglichkeiten offen: Eine Umformung der Regeln von Hand oder automatisiert mit einem Algorithmus. Die von uns in den Tests erzeugten DTDs lassen sich relativ einfach von Hand in eine nicht mehrdeutige Darstellung transformieren. Dies setzt allerdings eine vertiefte Kenntnis des SGML-Standards voraus. Alternativ dazu schlägt Ahonen [Aho96] einen Algorithmus vor, der eine kontextfreie Grammatik in eine SGML konforme, eindeutige Grammatik transformiert. Da uns aber keine Implementierung zur Verfügung stand, konnten diesbezüglich keine Tests durchgeführt werden.

8.2 Inferenz von Layoutmodellen

In Kapitel 1 wurden eine Reihe von Motiven für die Erkennung von Dokumenten aufgezählt. Ein mögliches davon ist, den Inhalt von Dokumenten, die in einer physischen Form vorliegen, zu editieren und wieder in einer physischen Form mit identischem Layout abzulegen. Für die Rücktransformation des editierten, logischen Dokuments wird daher neben dem logischen auch ein Layoutmodell benötigt.

Mit der Document Style Semantics and Specification Language (DSSSL) existiert seit wenigen Jahren ein Standard für die Transformation von SGML-Dokumenten in physische Repräsentationen. Um die in der Praxis auftretenden, zuweilen sehr komplexen typographischen Regeln adäquat beschreiben zu können, wurde mit *Scheme*, einem Lisp-Dialekt, eine komplette Programmiersprache in DSSSL integriert. In Tabelle 8.3 ist ein Ausschnitt eines DSSSL-Sourcecodes aufgelistet, der die Präsentation von Titeln und Absätzen eines Artikels definiert. Obgleich es sich um ein sehr einfaches Beispiel handelt, enthält das n-Gramm-Modell prinzipiell nicht genügend Information, um gleichartige Deklarationen zu erzeugen. Konstanten wie `*pIndent*` können nicht präzise deklariert werden, weil für das Modell alle relevanten Distanzen in wenige Klassen abgebildet werden. Ebenso ist unklar, auf welche Weise Berechnungen wie `(/ *titleFontSize* 2)` automatisch generiert werden können, da die Abhängigkeiten zwischen den Konstanten unbekannt sind. Gänzlich problematisch wird die automatisierte Erzeugung von DSSSL-Code, wenn statische Deklarationen wie in Tabelle 8.3 für die generische Beschreibung des Layouts nicht aus-

```
; Constants
(define *titleFontSize*         18pt)
(define *pFontSize*             (/ *titleFontSize* 2))
(define *pIndent*               3cm)
(define *pSpaceBefore*          .5cm)

; Elements
(element (ARTICLE TITLE)
        (make paragraph
                quadding:       'center
                font-size:      *titleFontSize*
                line-spacing:   *titleFontSize*
                font-weight:    'bold
                keep-with-next?:#t
                (process-children))

(element (P)
        (make paragraph
                font-size:      *pFontSize*
                line-spacing:   *pFontSize*
                start-indent:   *pIndent*
                space-before:   *pSpaceBefore*
                space-after:    *pSpaceBefore*
                (process-children))
```

Tabelle 8.3: Ausschnitt aus einer DSSSL-Spezifikation.

reichen, und daher die Generierung von Scheme-Code mit Kontrollstrukturen erforderlich macht.

Aufgrund der erwähnten Schwierigkeiten wurden im Zusammenhang mit der Inferenz von Layoutmodellen keine weiteren Nachforschungen angestrebt.

8.3 Modellinitialisierung aus Produktionsmodellen

Eine letzte Form der Integration mit dem SGML-Standard besteht in der Initialisierung des n-Gramm-Modells. Falls von einer Klasse zu erkennender Dokumente die DTD und die DSSSL-Spezifikation bekannt ist, so könnte daraus ein initiales Erkennungsmodell generiert werden. Dies ist insbesondere deshalb interessant, weil sich gezeigt hat, dass die Erstellung eines initialen Modells die

aufwendigste und fehlerträchtigste Phase in der Dokumenterkennung ist.

Eine Initialisierung könnte auf dem folgenden, naheliegenden Prinzip beruhen: Die logische Struktur kann durch Interpretation der Regeln in der DTD als generatives Modell erzeugt werden. Jedesmal, wenn in den Regeln mehrere Varianten auftreten (Abbruch einer Iteration, Wahl einer Option, Wahl einer Alternative) entscheidet der Zufall. Prinzipiell lassen sich so logische Strukturen erzeugen, die alle zur DTD konform sind.

Aus den logischen Zufallsdokumenten können weiter mit einem Layoutprozessor, der die DSSSL-Spezifikation umsetzt, die entsprechenden physischen Dokumente generiert werden. Beim automatisierten Lernen dieser künstlich erzeugten Dokumente stellen sich allerdings zwei wesentliche Probleme.

1. Für das Lernen der Dokumentstrukturen werden nicht nur die logische und physische Struktur benötigt, sondern auch die Verbindungen, die zwischen ihren Entitäten bestehen. Die Verbindungen zwischen den logischen und physischen Entitäten werden beim Setzen des Textes nur temporär innerhalb des Layoutprozessors hergestellt und sind von aussen nicht zugänglich.

   Eine mögliche Lösung dieses Problems besteht darin, an Stellen, in denen in der logischen Struktur beliebiger Text steht (`#PCDATA`) nicht zufälligen Text, sondern eindeutig codierte Marken als Text zu generieren. Da solche Textcodes im physischen Dokument weitgehend unverändert erscheinen, können sie mit dem logischen Dokument verglichen und zur Übereinstimmung gebracht werden. Daraus liesse sich eruieren, in welche physische Entitäten die logischen im Layoutprozess abgebildet wurden.

2. Eine DTD spezifiziert nur die Struktur von Dokumenten und macht keine Einschränkungen für deren Inhalt. Als Beispiel sollen die beiden Entitäten Titel und Absatz verglichen werden. In der Praxis bestehen Titel typischerweise aus wenigen Worten und äusserst selten aus mehr als einer Zeile, während Absätze typischerweise aus mehreren Zeilen bestehen. In einer DTD hingegen wird deren Inhalt üblicherweise als `#PCDATA` deklariert. Eine zufällige Generierung von Dokumenten würde daher ebenso häufig mehrzeilige Titel wie mehrzeilige Absätze erzeugen. Für ein automatisches Lernen der Strukturen ist es aber von Belang, aus wievielen Zeilen oder Wörtern die Entitäten bestehen, da sich dies direkt im n-Gramm-Modell widerspiegelt.

   Das Problem liesse sich dadurch lösen, dass `#PCDATA` Deklarationen mit Attributen angereichert würden, die statistische Verteilungen definieren, und somit die Anzahl der durchschnittlich generierten Zufallswörter beeinflussen.

Aus der obigen Diskussion folgt, dass eine Initialisierung des n-Gramm-Modells bei vorhandener DTD und DSSSL-Spezifikation grundsätzlich möglich sein sollte. Es muss allerdings erwartet werden, dass sich die Entwicklung und Implementierung der vorgeschlagenen Methode aufwendig und komplex gestaltet.

8.4 Diskussion

Die Integration des n-Gramm-Modells mit dem Dokumentproduktionsstandard SGML verspricht eine Reihe von Vorteilen, insbesondere aufgrund der Vielzahl von Tools, die für SGML verfügbar sind. Am intensivsten wurde von uns die Möglichkeit untersucht, durch induktive Inferenz aus dem n-Gramm-Modell eine DTD zu generieren. Dazu wurde ein Algorithmus implementiert, der eine initiale, sehr generelle, kontextfreie Grammatik erzeugt, die durch gezielte Interpretation von n-Grammen schrittweise spezialisiert wird. Die Versuche haben gezeigt, dass dieses System DTDs erzeugt, die relativ wenige Unterschiede zu handgenerierten DTDs aufweisen. Dennoch ist im allgemeinen eine manuelle Nachbearbeitung der erzeugten DTDs notwendig, einerseits, um sie leserlicher zu gestalten, andererseits, um Mehrdeutigkeiten zu beheben.

Die Inferenz eines typographischen Modells aus dem n-Gramm-Modell birgt grundlegende Probleme und wird erst in zukünftigen Forschungen näher untersucht werden. Die Initialisierung eines n-Gramm-Modells aus einer vorhandenen DTD und DSSSL-Spezifikation scheint uns möglich, aber aufwendig. Eine Implementierung dieses Ansatzes existiert noch nicht.

Kapitel 9

Erweiterungen

Auf den Erfahrungen aufbauend, die mit dem n-Gramm basierten Ansatz für die Dokumenterkennung gemacht wurden, soll in diesem Kapitel der Blick auf seine zukünftig mögliche Entwicklung gerichtet werden.

Der Ansatz wurde bisher an Dokumenten getestet, die wissenschaftlichen Artikeln ähnlich sind. Künftige Tests werden zeigen müssen, wie weit andere Dokumenttypen wie Zeitungsartikel oder Briefköpfe erkennbar sind. Von besonderem Interesse kann zudem die Anwendung des Ansatzes auf die Mikrostruktur von Dokumenten sein. Wir gehen davon aus, dass für die Erkennung der Mikrostruktur weder das Modell noch der Erkennungsalgorithmus angepasst werden müssen. Die einzige Modifikation wird in der Klassifizierung der physischen Attribute in physische Entitäten nötig sein. So wäre beispielsweise eine Feinklassifizierung von Wörtern nicht nur nach dem Font, sondern zusätzlich nach ihrem Textinhalt in Sonderzeichen, Zahlen, normale Wörter und Schlüsselwörter, für die Erkennung der Mikrostruktur unabdingbar.

Die folgenden Abschnitte diskutieren Erweiterungsmöglichkeiten sowie Modells und des Erkennungsalgorithmus.

9.1 Erweiterungen des Modells

Grundsätzlich ist es denkbar, das n-Gramm-Modell mit klassischen Modellen der Dokumenterkennung zu kombinieren, beispielsweise Fuzzy Logik, Prologregeln oder Grammatiken. Man sollte sich allerdings vor einer überstürzten Einführung neuer Ansätze hüten, wenn sie nur einen begrenzten Erfolg versprechen. Die Festlegung auf die n-Gramm-Wahrscheinlichkeiten als einziges Konzept mag primitiv anmuten. Es hat jedoch die Eigenschaft, leicht und intuitiv verständlich zu sein, was vor allem für die Implementierung des Ansatzes von unschätzbarem

Vorteil ist. Zudem konnte in den Experimenten gezeigt werden, dass das Modell die getesteten Dokumentstrukturen vollständig zu beschreiben vermag.

Wenn dennoch eine Erweiterung des Modells erwogen wird, beispielsweise um komplexere Strukturen darzustellen, sollten daher vorerst die Möglichkeiten des n-Gramm-Ansatzes voll ausgeschöpft werden.

Selbst wenn man an der Grundidee festhält, Nachbarschaftsbeziehungen zu repräsentieren, so liessen sich a priori eine grosse Menge neuer n-Gramme definieren. Die Einführung jedes neuen n-Gramms ist aber nur dann sinnvoll, wenn es im Algorithmus für die Konstruktion der logischen Struktur auch explizit verwendet wird. Da sich eine solche Integration in den Algorithmus unter Umständen sehr aufwendig gestalten kann, sollten neue n-Gramme möglichst sorgfältig ausgewählt werden. Als Mass für die Qualität von n-Grammen kann deren Entropie bestimmt werden.

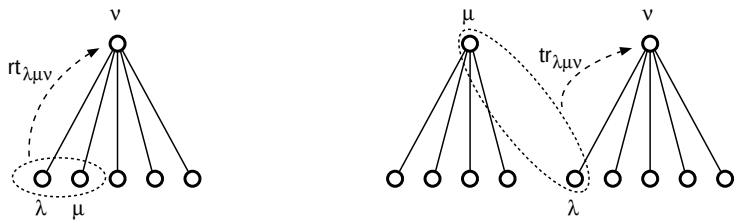


Abbildung 9.1: Alternative Typen von n-Grammen.

Ein weiteres wichtiges Qualitätskriterium für n-Gramme ist deren Nützlichkeit beim Aufbau der logischen Struktur. So beträgt die Entropie der n-Gramme $b_{\lambda\mu}$ und $b_{\lambda\mu\nu}$ in „LinuxDoc“ Dokumenten zwar 0; trotzdem dienen sie dem Aufbau der logischen Struktur relativ wenig, weil sie bottom-up konstruiert wird. Die n-Gramme $b_{\lambda\mu}$ und $b_{\lambda\mu\nu}$ ermöglichen jedoch nur top-down Voraussagen von Entitäten. Insbesondere im Hinblick auf die Unterstützung des Konstruktionsprozesses bottom-up und von links nach rechts könnten die folgenden Trigramme sein, die auch in Abbildung 9.1 illustriert sind. Das n-Gramm

$$rt_{\lambda\mu\nu} = P(x = \nu \mid R(\lambda\mu) \wedge T(\lambda\nu))$$

beschreibt die Wahrscheinlichkeit, dass ein Knoten mit Label λ als Vorgängerknoten ν hat, wenn sein rechter Nachbar das Label μ trägt. Das n-Gramm

$$tr_{\lambda\mu\nu} = P(x = \nu \mid T(\lambda\nu) \wedge R(\mu\nu))$$

beschreibt die Wahrscheinlichkeit, dass ein Knoten mit Label λ als Vorgängerknoten ν hat, wenn ν der rechte Nachbar eines Knotens mit Label μ ist.

Es ist anzunehmen, dass solche geschickt gewählten Nachbarschaftsbeziehungen, ausgedrückt durch n-Gramme, den Konstruktionsalgorithmus sowohl sicherer als auch einfacher machen.

9.2 Erweiterungen des Algorithmus

Wesentlich stärker als beim Modell ist beim Konstruktionsalgorithmus ein Entwicklungspotential zu vermuten, da er die am wenigsten stabilisierte Komponente des Ansatzes darstellt. Diese Annahme stützt sich auf die Tatsache, dass im Laufe der Entwicklung des vorgestellten Ansatzes der Algorithmus mehrere Male fundamental verändert werden musste, während das Modell praktisch unangetastet blieb. Die folgenden Aspekte könnten Gegenstand künftiger Forschungen im Zusammenhang mit dem Algorithmus sein.

Fehlertoleranz: Die gegenwärtig kaum vorhandene Fehlertoleranz ist eine Folge des Algorithmus und nicht des Modells. Die schrittweise Gruppierung von Knoten fehlertoleranter zu machen heisst, auch Gruppierungsvarianten zuzulassen, die nur einen Teil der Gruppierungsbedingungen erfüllen. Dies hat zur Konsequenz, dass damit von jeder partiellen Lösung zusätzliche Alternativen ausgehen und sich der Suchbaum verbreitert. Die fehlertolerantere Konstruktion wird daher nur auf Kosten der Effizienz zu erreichen sein.

Suchstrategie: Einer Vergrösserung des Suchbaums, infolge zusätzlicher Fehlertoleranz, kann mit optimierten Suchstrategien begegnet werden, beispielsweise A^*.

Vorrangige Interpretationsstellen: Der aktuelle Algorithmus konstruiert die logische Struktur konsequent bottom-up von links nach rechts. Besser wäre es hingegen, die logische Struktur an Stellen der physischen Struktur zu konstruieren, die aufgrund charakteristischer, physischer Attribute mit grosser Gewissheit interpretierbar sind. Vorrangig darauf konstruierte, sichere Strukturfragmente liefern zusätzliche Kontextinformation, welche die Konstruktion an den unsicheren Stellen der physischen Struktur vereinfachen können.

Split in Teilprobleme: Die Aufteilung in Teilprobleme wird wesentlich vom Parameter *splitdepth* beeinflusst, der seinerseits von der Dokumentklasse abhängig ist. Durch Analyse des Dokumentmodells könnte dieser Parameter automatisch bestimmt werden. Ferner könnte die Aufteilung in Teilprobleme verallgemeinert werden, so dass sie nicht nur einmal, sondern mehrmals rekursiv auf Teilprobleme anwendbar bleibt.

Einstellbarer Erkennungsmodus: Das Modell bietet die Möglichkeit, durch Variation des Generalized-Means-Parameters α die Strenge der Modellkonformitätsmessung zu beeinflussen. Damit lässt sie sich stufenlos zwischen fehlertolerantem und -intolerantem Verhalten einstellen. Auf der Ebene des Algorithmus fehlt diese Möglichkeit. So sollte sich das System in einer beaufsichtigten Lernphase gegenüber Fehlern sehr intolerant verhalten, um möglichst viel korrekte Information über die Dokumentklasse zu sammeln. Demgegenüber muss dem Benutzer die Möglichkeit offen stehen, die Fehlertoleranz zu erhöhen, um die Dokumenterkennung unbeaufsichtigt als Batch-Prozess ablaufen zu lassen.

Kapitel 10

Zusammenfassung

Seit jeher spielen Dokumente in der Verwaltung von Information eine zentrale Rolle. Dabei hat die Automatisierung der Informationsverarbeitung deren allgemeine Bedeutung zusätzlich verstärkt.

Ein verbreitetes Problem ist die Transformation von Dokumenten zwischen verschiedenen Medien. Selbst Transformationen ohne Medienbrüche zwischen unterschiedlichen Formaten sind in der Praxis trotz grosser Fortschritte in der Standardisierung elektronischer Dokumentformate problematisch. Als mögliche Lösung bietet sich hier die Dokumenterkennung an. Sie bezeichnet den Prozess, in dem Elemente eines Dokumentbildes in abstrakte Darstellungen von Buchstaben, Schriftarten, Layout und logischer Struktur zurückübersetzt werden.

Die Forschung der Dokumenterkennung konzentrierte sich in den letzten zwei Dekaden vor allem auf die optische Zeichenerkennung und erst in jüngerer Zeit auf die Erkennung von Dokumentstrukturen. Wenig Beachtung wurde dabei der Frage geschenkt, auf welche Weise ein Anwender in den üblicherweise langwierigen und komplexen Prozess der Erkennung einbezogen werden kann. Da man davon ausgehen muss, dass es in absehbarer Zeit keine vollautomatischen Dokumenterkennungssysteme geben wird, müssen interaktive Systeme geschaffen werden, die in einer angenehmen Weise mit dem Anwender kooperieren. Das System darf einen Anwender nicht zu simplen Korrekturen missbrauchen, sondern es soll ihn in den Erkennungsprozess mit einbeziehen. Begeht es Fehler, so müssen Benutzerkorrekturen so schnell wie möglich in das Erkennungsmodell integriert werden, um Wiederholungen zu vermeiden.

Das Hauptziel dieser Arbeit ist daher die Entwicklung eines Ansatzes für die Erkennung der logischen Dokumentstruktur mit einem interaktiven System, das sich schrittweise an das Problem anpasst und somit seine Leistung kontinuierlich verbessert.

Da sich die Lernfähigkeit in klassische Ansätze der Dokumenterkennung nur schwer integrieren lässt, wurde ein neuartiges Modell entwickelt. Es basiert auf der statistischen Repräsentation lokaler Muster in Dokumentstrukturen mit n-Grammen. Das n-Gramm-Modell bietet die Möglichkeit, auf einfache Weise Dokumentstrukturen lernen zu können und Zusatzinformationen zu integrieren. Es ermöglicht ferner ein effizientes initiales und inkrementelles Lernen. Als Mass für die Trennschärfe verschiedener Typen von n-Grammen dient die Entropie. Mit ihr lässt sich die Qualität eines gelernten Modells abschätzen noch bevor es für die Erkennung verwendet wird.

Passend zum Modell wurde ein Erkennungsalgorithmus entwickelt. Auf einer heuristischen Methode basierend konstruiert er schrittweise die logische Struktur eines Dokuments, sofern dessen physische Struktur bekannt ist. In jedem Konstruktionsschritt werden verschiedene, sich gegenseitig konkurrierende Alternativen entwickelt. Durch Bewertung der Alternativen und unter Verwendung einer best-first Strategie wird eine optimale Lösung gesucht. Die Bewertung der Alternativen erfolgt mit der Generalized-Means-Funktion, einer aus der Verarbeitung unsicherer Information bekannten allgemeinen Aggregationsfunktion.

Der Ansatz wurde an einer Reihe von verschiedenen, artikelähnlichen Dokumentklassen getestet. Die Haupttests wurden an 26 Dokumenten mittlerer Komplexität durchgeführt, die insgesamt 390 Seiten umfassen und aus idealen, von PostScript-Dokumenten erzeugten Bildern bestanden. Sind bei der Erkennung der ersten Dokumente noch zirka eineinhalb Korrekturen, beziehungsweise inkrementelle Lernschritte pro Seite nötig, so reduziert sich die Rate nach zwanzig Dokumenten auf etwa eine Korrektur alle vier Seiten. Parallel dazu ist eine Konvergenz des Modells zu beobachten. Trotz der theoretisch exponentiellen Komplexität des Erkennungsalgorithmus, ist empirisch ein linearer Zusammenhang zwischen Problemgrösse und Anzahl Erkennungsschritten festzustellen. Mit einem vollständigen Modell benötigt die Erkennung auch bei grösseren Dokumenten von dreissig Seiten nur wenige Sekunden. Als Nachteil des Algorithmus ist die hohe Empfindlichkeit auf Fehler in der physischen Struktur beobachtet worden. Er weist in seiner aktuellen Form eine nur marginale Fehlertoleranz auf.

Schliesslich wurden Möglichkeiten aufgezeigt, wie sich der n-Gramm-Ansatz in Standards der Dokumentproduktion integrieren lässt. Dazu wurde ein Algorithmus entwickelt, der die statistischen Informationen der n-Gramme in eine kontextfreie Grammatik in Form einer DTD transformiert. Da die generierten Regeln im allgemeinen mehrdeutig sind, ist eine leichte manuelle Nachbearbeitung notwendig, um eine SGML konforme DTD zu erhalten. Eine Initialisierung der n-Gramme aus einer DTD zusammen mit einem typographischen Modell wurde nicht entwickelt, scheint jedoch realisierbar.

In dieser Arbeit wurde versucht, mit einem neuartigen Modell einen Beitrag zur Bewältigung der Problematik der Dokumenterkennung zu leisten. Mit der statistischen Modellierung von Dokumentstrukturen mit n-Grammen konnten vielversprechende Erkenntnisse gewonnen werden, die zu weiteren Forschungen in diesem Bereich ermutigen.

Literaturverzeichnis

[Ado86] Adobe Systems Incorporated. *PostScript Language.* Addison-Wesley, 1986.

[Ado93] Adobe Systems Incorporated. *Portable Document Format Reference Manual.* Addison-Wesley, 1993.

[Aho96] Helena Ahonen. *Generating Grammars for Structured Documents Using Grammatical Inference Methods.* PhD thesis, University of Helsinki, Finland, November 1996.

[Aki95a] Tunde Akindele. Construction of generic models of document structures using inference of tree grammars. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1995.

[Aki95b] Tunde Akindele. *Vers un système de construction automatique de modèles génériques de documents.* PhD thesis, CRIN-Nancy, 1995.

[Ald92] Aldus Corporation. Tiff revision 6.0. ftp://ftp.sgi.com/graphics/tiff/TIFF6.ps.Z, 1992.

[AM87] T. Akiyama and I. Masuda. A method for document image segmentation based on projection profiles, stroke densities and circumscribed rectangles. In *Systems and Computers in Japan.* 1987.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers. Principles, Techniques and Tools.* Addison-Wesley, 1986.

[Bai87] Henry S. Baird. The skew angle of printed documents. In Society of Photographic Scientists and Engineers, editors, *Proceedings of the Conference on Hybrid Imaging Systems*, Rochester, 1987.

[Bap98a] Frédéric Bapst. *Reconnaissance de documents assisté: architecture logicielle et intégration de savoir-faire*. PhD thesis no. 1228, University of Fribourg, 1998.

[BAP98b] Hans-Jörg Bullinger, Christoph Altenhofen, and Mirjana Petrovic. Der Umgang mit virtuellen Papierbergen. *Computerworld Schweiz*, 98(33), August 1998.

[Bay93] Thomas A. Bayer. Understanding structured text documents by a model based document analysis system. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1993.

[BBI95] Frédéric Bapst, Rolf Brugger, and Rolf Ingold. Towards an interactive document recognition system. Internal working paper 95-09, IIUF-Université de Fribourg, March 1995.

[BBI98] Rolf Brugger, Frédéric Bapst, and Rolf Ingold. A DTD extension for document structure recognition. In R. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography (EP-RIDT)*, Lecture Notes in Computer Science. Spinger, 1998.

[BBMS94] Thomas A. Bayer, Ulrich Bohnacker, and Heike Mogg-Schneider. InfoPortLab — an experimental document understanding system. In *Document Analysis Systems (DAS)*, 1994.

[BBZI96a] Frédéric Bapst, Rolf Brugger, Abdelwahab Zramdini, and Rolf Ingold. Integrated multi-agent architecture for assisted recognition. In *Document Analysis Systems (DAS)*, 1996.

[BBZI96b] Frédéric Bapst, Rolf Brugger, Abdelwahab Zramdini, and Rolf Ingold. L'intégration des données dans un système de reconnaissance de documents assistée. In *CNED*, 1996.

[BI98a] Frédéric Bapst and Rolf Ingold. Using typography in document image analysis. In R. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography (EP-RIDT)*, Lecture Notes in Computer Science. Spinger, 1998.

[BI98b] Frédéric Bapst and Rolf Ingold. Using typography in document image analysis. In R. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography (EP-RIDT)*, Lecture Notes in Computer Science. Springer, 1998.

[BK97] Frédéric Bapst and Oliver Krone. A coordination kernel for coupling heterogeneous programming environments. Internal working paper n. 97-03, Université de Fribourg, 1997.

[BML95] Andrew Birrell, Paul McJones, and Russell Lang. pstotext. http://www.research.digital.com/SRC/virtualpaper/, 1995.

[Bol96] Daniel Bollinger. Inferenz und Spezialisierung kontextfreier Regeln mit statistischen Zusatzinformationen. master's thesis report in computer science, Uni Fribourg, 1996.

[Bun98] Bundesamt für Statistik. BFS, Produktions- Auftrags-, Umsatz- und Lagerstatistik. Technical report, Bundesamt für Statistik, Bern, 1998. http://www.admin.ch/bfs/stat_ch/ber06/dufr06.htm.

[BW95] Thomas A. Bayer and Hanno Walischewsky. Experiments on extracting structural information from paper documents using syntactic pattern analysis. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1995.

[BW97] H. Bunke and P. S. P. Wang. *Handbook of Optical Character Recognition and Document Analysis.* World Scientific Publishing Company, 1997.

[BZI93] Frédéric Bapst, Abdelwahab Zramdini, and Rolf Ingold. A scenario model advocating user-driven adaptive recognition systems. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1993.

[BZI97] Rolf Brugger, Abdelwahab Zramdini, and Rolf Ingold. Modeling documents for structure recognition using generalized n-grams. In *Fouth International Conference on Document Analysis and Recognition (ICDAR).* IEEE, 1997.

[Cha93] Eugene Charniak. *Statistical language learning.* MIT Press, 1993.

[Che93] Yannick Chenevoy. *Reconnaissance structurelle de documents imprimées: études et réalisations.* PhD thesis, CRIN-Nancy, 1993.

[Cla99] James Clark. SP - an SGML system conforming to international standard ISO 8879 – standard generalized markup language. http://www.jclark.com/sp/, 1999.

[Con93] Alan Conway. Page grammars and page parsing, a syntactic approach to document layout recognition. In *Proceedings of the International*

Conference on Document Analysis and Recognition (ICDAR), pages 761–764, 1993.

[Cov99] Robin Cover. The SGML/XML Web Page. http://www.oasis-open.org/cover/, 1999.

[DBH+94] Andreas Dengel, R. Bleisinger, Rainer Hoch, F. Hönes, and M. Malberg. OFFICEMAID — a system for office mail analysis, interpretation and delivery. In *Document Analysis Systems (DAS)*, 1994.

[DDS+97] Dov Dori, David Doermann, Christian Shin, Robert Haralick, Ishin Phillips, Mitchell Buchmann, and David Ross. The representation of document structure: A generic object-process analysis. In H. Bunke and P. S. P. Wang, editors, *Handbook of Optical Character Recognition and Document Analysis*. World Scientific, 1997.

[Den92] Andreas Dengel. ANASTASIL: A System for Low-Level and High-Level Geometric Analysis of Printed Documents. In Henry S. Baird, Horst Bunke, and Kazuhiko Yamamoto, editors, *Structured Document Image Analysis*. Springer-Verlag, 1992.

[Den93] Andreas Dengel. Initial learning of document structure. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1993.

[Doe97] David Doermann. The retrieval of document images: A brief survey. In *International Conference on Document Analysis and Recognition (ICDAR)*, 1997.

[EMS93] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. Automated acquisition of rules for document understanding. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1993.

[EMS95] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A knowledge based approach to the layout analysis. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1995.

[FX93] Peter Fankhauser and Yi Xu. Markitup! an incremental approach to document structure recognition. *Electronic Publishing*, December 1993.

[Gho99] Ghostscript, Ghostview and GSview Homepage. http://www.cs.wisc.edu/~ghost/, 1999.

[Han93] D. J. Hand. *Artificial Intelligence, Frontiers in Statistics.* Chapman & Hall, 1993.

[Her92] Joachim Hertzberg. The QWERTZ project. http://www-fit-ki.gmd.de/projects/qwertz.html, 1992.

[HFD90] Stuart C. Hinds, James L. Fisher, and Donald P. D'Amato. A document skew detection method using run-length encoding and the hough transform. In *Proceedings of the 10th International Conference on Pattern Recognition.* IEEE, 1990.

[HT98] Jonathan J. Hull and Suzanne L. Taylor. *Document Analysis Systems.* World Scientific, 1998.

[Hu94] Tao Hu. *New methods for robust and efficient recognition of the logical structures in documents.* PhD thesis, IIUF-Université de Fribourg, 1994. PhD Thesis no. 1076.

[ICD95] *International Conference on Document Analysis and Recognition (ICDAR).* IEEE Computer Society Press, 1995.

[ICD97] *International Conference on Document Analysis and Recognition (ICDAR).* IEEE Computer Society Press, 1997.

[Ing88] Rolf Ingold. *Une nouvelle approche de la lecture optique intégrant la reconnaissance des structures de documents.* PhD thesis no. 777, EPFL, Lausanne, 1988.

[Ing91] Rolf Ingold. A document description language to drive document analysis. In *International Conference on Document Analysis and Recognition (ICDAR)*, 1991.

[ISO86a] ISO. Information processing — text and office systems — office document architecture (ODA) and interchange format. (ISO 8613). Geneva: ISO, 1986.

[ISO86b] ISO. Information processing — text and office systems — standard generalized markup language (SGML) (ISO 8879). Geneva: ISO, 1986.

[ISO96] ISO. Document style semantics and specification language (DSSSL) (ISO 10179). Geneva: ISO, 1996.

[KF92] George J. Klir and Tina A. Folger. *Fuzzy Sets, Uncertainty, and Information.* Prentice-Hall International, 1992.

[Knu86] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, 1986.

[Kre93] Joachim Kreich. Robust recognition of documents. In *International Conference on Document Analysis and Recognition (ICDAR)*, 1993.

[Lan98] Barbara Lange. Workflow-Management-Systeme, Integration von WWW und Groupware. *iX*, 98(4), 98.

[Lin99] The linux documentation project homepage. http://metalab.unc.edu/LDP/, 1999.

[Lug94] George F. Luger. *Cognitive Science*. Academic Press, 1994.

[LZ98] Daniel Lopresti and Jiangying Zhou. Document analysis and the world wide web. In Jonathan Hull and Suzanne Taylor, editors, *Document Analysis Systems II*. World Scientific, 1998.

[Mil98] Udo Miletzki. Document on the move: DA&IR driven mail piece processing today and tomorrow. In Jonathan Hull and Suzanne Taylor, editors, *Document Analysis Systems II*. World Scientific, 1998.

[NS95] Debashish Niyogi and Sargur N. Srihari. Knowledge-based derivation of document logical structure. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1995.

[NSV97] George Nagy, Sharad Seth, and Mahesh Viswanathan. DIA, OCR and the WWW. In H. Bunke and P. S. P. Wang, editors, *Handbook of Optical Character Recognition and Document Analysis*. World Scientific, 1997.

[OK95] Lawrence O'Gorman and Rangachar Kasturi. *Document Image Analysis*. IEEE Computer Society Press, 1995.

[Pae89] A. W. Paeth. A fast algorithm for general raster rotation. In Canadian Information Processing Society, editor, *Proceedings of the Conference on Graphics Interfaces*, 1989.

[Pea85a] Judea Pearl. *Heuristics — Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1985.

[Pea85b] Judea Pearl. *Heuristics — Intelligent Search Strategies for Computer Problem Solving*, chapter Basic Heuristic-Search Procedures, pages 33–72. Addison-Wesley, 1985.

[Rab90] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in Speech Recognition*, pages 267–296. Morgan Kaufmann Publishers, 1990.

[RAF95] RAF Technology, Inc. *DAFS Library, Programmer's Guide and Reference*, 1995.

[Ree95] Colin R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill International, 1995.

[Rei89] Brian Reid. Electronic mail of structured documents. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents.* Cambridge University Press, 1989.

[RI98] Lyse Robadey and Rolf Ingold. Détection automatique de pages vides dans un système d'archivage: premières experiences. In *CIFED*, 1998.

[SGM99] The SGMLTools project homepage. http://www.sgmltools.org/, 1999.

[Sha95] Keith Shafer. Creating DTDs via the gb-engine and fred. Technical report, OCLC Online Computer Library Center, 1995.

[SMBK95] P. Suda, G. Maderlechner, H. Bock, and H. P. Klunder. How can document analysis help in capturing five million pages? In *International Conference on Document Analysis and Recognition (ICDAR)*, 1995.

[SP98] Mirjana Stanisic-Petrovic. Der Dokument Management Systeme Markt. *Computerworld*, 98(45), 98.

[TL96] Suzanne L. Taylor and Mark Lipshutz. Document understanding for multiple document representations. In *Document Analysis Systems II.* World Scientific, 1996.

[WCW82] K. Y. Wong, R. G. Casey, and F. H. Whal. Document analysis systems. Technical report, IBM, 1982.

[Win84] Patrick H. Winston. *Artificial Intelligence.* Addison-Wesley, second edition, 1984.

[ZL97] Jiangying Zhou and Daniel Lopresti. Extracting text from WWW images. In *International Conference on Document Analysis and Recognition (ICDAR)*, 1997.

[Zra95] Abdelwahab Zramdini. *Study of Optical Font Recognition Based on Global Typographical Features.* PhD thesis no. 1106, University of Fribourg, 1995.

Anhang A

Testdaten

Auf den folgenden Seiten ist die vereinfachte Document Type Definition (DTD) der „LinuxDoc“ Dokumente aufgeführt. Nicht alle darin vorkommenden Tags gelangen in den „LinuxDoc“ Dokumenten auch tatsächlich zur Anwendung, da der Textprozessor *SGMLTools* keine graphischen Elemente erlaubt.

Anschliessend an die DTD sind einige Auszüge aus LinuxDoc-Dokumenten dargestellt.

```
<!element linuxdoc o o
        (sect | chapt | article | report |
 book | letter | telefax | slides | notes | manpage ) >

<!element  p o o (( %inline | %sectpar )+) +(newline) >
<!element dp o o (( %inline | %sectpar )+) +(newline) >
<!element em - - (%inline)>
<!element bf - - (%inline)>
<!element it - - (%inline)>
<!element sf - - (%inline)>
<!element sl - - (%inline)>
<!element tt - - (%inline)>
<!element sq - - (%inline)>
<!element cparam - - (%inline)>
<!element quote - - ((%inline; | %sectpar;)*, p*)+ >
<!element tscreen - - ((%inline; | %sectpar;)*, p*)+ >
<!element itemize - - (item+)>
<!element enumerate - - (enum+)>
<!element list - - (item+)>
<!element descrip - - (tag?, p+)+ >
```

```
<!element item o o ((%inline; | %sectpar;)*, dp*) >
<!element enum o o ((%inline; | %sectpar;)*, dp*) >
<!element tag - o (%inline)>
<!element code - - rcdata>
<!element verb - - rcdata>
<!element label - o empty>
<!element ref - o empty>
<!element url - o empty>
<!element htmlurl - o empty>

<!element article - -
        (titlepag, header?,
         toc?, lof?, lot?, p*, sect*,
         (appendix, sect+)?, biblio?) +(footnote)>

<!element author - o (name, thanks?, inst?,
                      (and, name, thanks?, inst?)*)>
<!element name o o (%inline) +(newline)>
<!element date - o (#pcdata) >
<!element titlepag o o (title, author, date?, abstract?)>
<!element title - o (%inline, subtitle?) +(newline)>
<!element subtitle - o (%inline)>
<!element newline - o empty >
<!element abstract - o (%inline)>
<!element toc - o empty>
<!element heading o o (%inline)>
<!element chapt - o (%sect, sect*) +(footnote)>
<!element sect  - o (%sect, sect1*) +(footnote)>
<!element sect1 - o (%sect, sect2*)>
<!element sect2 - o (%sect, sect3*)>
<!element sect3 - o (%sect, sect4*)>
<!element sect4 - o (%sect)>
<!element appendix - o empty >
<!element footnote - - (%inline)>
<!element file - - (#pcdata)>

<!element idx - - (#pcdata)>
<!element cdx - - (#pcdata)>
<!element nidx - - (#pcdata)>
<!element ncdx - - (#pcdata)>
```

```
<!-- entities, attlists, maps, ... -->
<!usemap oneline opening>

<!usemap oneline (from,to)>

<!entity % emph
" em|it|bf|sf|sl|tt|cparam " >

<!entity % index "idx|cdx|nidx|ncdx" >

<!entity % xref " label|ref|pageref|cite|url|htmlurl|ncite " >

<!entity % inline
" (#pcdata | f| x| %emph; |sq| %xref | %index | file )* " >

<!entity % list
        " list | itemize | enum | descrip " >

<!entity % par
"  %list; | comment | lq | quote | tscreen " >

<!entity % mathpar " dm | eq " >

<!entity % thrm
        " def | prop | lemma | coroll | proof | theorem " >

<!entity % litprog " code | verb " >

<!entity % sectpar
        " %par; | figure | tabular | table | %mathpar; |
  %thrm; | %litprog; ">

<!entity % isoent system "isoent">
%isoent;

<!entity urlnam sdata "urlnam" >
<!entity refnam sdata "refnam" >
<!entity tex sdata "[tex   ]" >
<!entity latex sdata "[latex ]" >
<!entity latexe sdata "[latexe]" >
```

```
<!entity tm     sdata "[trade ]" >
<!entity dquot  sdata "[quot  ]" >
<!entity ero    sdata "[amp   ]" >
<!entity etago '</' >
<!entity   Ae  '&Auml;' >
<!entity   ae  '&auml;' >
<!entity   Oe  '&Ouml;' >
<!entity   oe  '&ouml;' >
<!entity   Ue  '&Uuml;' >
<!entity   ue  '&uuml;' >
<!entity   sz  '&szlig;' >

<!entity ptag '<p>' >
<!entity psplit '</p><p>' >

<!shortref pmap
"&#RS;B" null
"&#RS;B&#RE;" psplit
"&#RS;&#RE;" psplit
--'"' qtag  --
       "[" lsqb
       "~" nbsp
       "_" lowbar
       "#" num
       "%" percnt
       "^" circ
       "{" lcub
       "}" rcub
       "|" verbar >

<!usemap pmap p>

<!shortref desmap
        "&#RS;B" null
        "&#RS;B&#RE;" ptag
        "&#RS;&#RE;" ptag
        "~" nbsp
        "_" lowbar
        "#" num
        "%" percnt
        "^" circ
```

```
        "[" lsqb
        "]" rsqb
        "{" lcub
        "}" rcub
        "|" verbar >
<!usemap desmap tag>
<!usemap desmap descrip>

<!usemap global (list,itemize,enumerate)>
<!entity space " ">
<!entity null "">

<!attlist eps
        file cdata #required
        height cdata "5cm"
        angle cdata "0">
<!attlist figure
loc cdata "tbp"
caption cdata "Caption">
<!attlist ph
        vspace cdata #required>
<!attlist img
src cdata #required>

<!usemap global (rf,phr)>
<!usemap global (def,prop,lemma,coroll,proof,theorem)>
<!usemap oneline thtag>
<!entity   qtag     '<sq>' >

<!shortref global
      "&#RS;B" null  -- delete leading blanks --
  --    '"' qtag --
      "[" ftag
      "~" nbsp
      "_" lowbar
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>
```

```
<!usemap global linuxdoc>
<!attlist label id cdata #required>

<!-- ref modified to have an optional name field HG -->
<!attlist ref
        id cdata #required
        name cdata "&refnam">

<!-- url entity added to have direct url references HG -->
<!attlist url
        url cdata #required
        name cdata "&urlnam" >

<!-- htmlurl entity added to have quieter url references esr -->
<!attlist htmlurl
        url cdata #required
        name cdata "&urlnam" >

<!attlist pageref
        id cdata #required>
<!usemap   #empty   x >

<!-- Hacked by mdw to exclude abstract;
     abstract now part of titlepag -->

<!attlist article
        opts cdata "null">
<!attlist report
        opts cdata "null">

<!attlist book
        opts cdata "null">
<!usemap oneline titlepag>
<!usemap global thanks>
<!entity nl "<newline>">

<!-- Hacked by mdw -->
<!usemap oneline abstract>

<!usemap global footnote>
```

```
<!usemap oneline (chapt,sect,sect1,sect2,sect3,sect4)>
<!entity % sect "heading, header?, p* " >
<!attlist cite
        id cdata #required>

<!attlist ncite
        id cdata #required
note cdata #required>

<!attlist biblio
style cdata "linuxdoc"
        files cdata "">

<!attlist slides
opts cdata "null">
<!entity  % addr "(address?, email?, phone?, fax?)" >


<!attlist letter
opts cdata "null">
```

The Linux HOWTO Index

by Tim Bynum, *linux-howto@sunsite.unc.edu* v2.10.84, 4 August 1998

This document contains an index to the Linux HOWTOs and mini-HOWTOs, as well as other information about the HOWTO project.

Contents

1 What Are Linux HOWTOs?

Linux HOWTOs are documents which describe in detail a certain aspect of configuring or using Linux. For example, there is the *Installation HOWTO*, which gives instructions on installing Linux, and the *Mail HOWTO*, which describes how to set up and configure mail under Linux. Other examples include the *NET-3 HOWTO* and the *Printing HOWTO*.

HOWTOs are comprehensive docs - much like an FAQ but generally not in question-and-answer format. However, many HOWTOs contain an FAQ section at the end.

There are several HOWTO formats available: plain text, PostScript, DVI, and HTML.

In addition to the HOWTOs, there are a multitude of mini-HOWTOs on short, specific subjects. They are only available in plain text and HTML format.

2 Where Can I Get Linux HOWTOs?

HOWTOs can be retrieved via anonymous FTP from the following sites:

- *ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO*
- *ftp://tsx-11.mit.edu/pub/linux/docs/HOWTO*

Abbildung A.1: Eine typische Titelseite mit Abstract und Inhaltsverzeichnis.

Put the following lines in /etc/rc.d/rc.local:

```
-----
#!/bin/sh
#       Put any local setup commands in here
#
INITTY=/dev/tty[1-6]
PATH=/sbin:/etc:/bin:/usr/sbin:/usr/bin
#
#       kbd - Set the the console font and keyboard
#       set numlock and set metabit mode on tty1 .. tty8
for tty in $INITTY
do
#       setleds -D +num < $tty > /dev/null
        setmetamode metabit < $tty > /dev/null
done
#       Latin8(Hebrew) keyboard/console
setfont iso08.f16
mapscrn trivial
loadkeys Hebrew
#       enable mapping
for tty in $INITTY
do
        echo -n -e "\\033(K" >$tty
done
-----
```

NOTE: If you are using X Windows be careful with "setleds", it may hang the X server.

The above setup works fine with the Hebrew version of pico (pine) and displays correctly ISO 8859-8 Hebrew (X Windows, MS Windows).

4 X Windows setup - XFree86 3.1

4.1 Hebrew fonts.

XFree86 3.1 comes with two Hebrew fonts: heb6x13, heb8x13. Additional Hebrew fonts can be found on the Net:

- The web Type1 fonts (Helvetica/David style (proportional) and Courier/Shalom Stick style (fixed space)) from the snunit-project archive at *ftp://snunit.huji.ac.il/pub/fonts/* <ftp://snunit.huji.ac.il/pub/fonts/>, it's good for netscape Hebrew pages.
- Avner Lottem, (*lottem@techUnix.technion.ac.il*) put some Hebrew-ISO 8859-8 fonts on archive at *ftp://sunsite.unc.edu/pub/Linux/X11/fonts/hebxfonts-0.1.tgz* <ftp://sunsite.unc.edu/pub/Linux/X11/fonts/>, it has a font that's good for dosemu under X-Windows (read his README file).

4.2 Installing fonts

- Fonts exaptable: pcf (Portable Compiled Format), bdf (Bitmap Distribution Format), pfb (Type1 fonts).

Abbildung A.2: Beispiel einer Seite mit vorformatiertem Text, einzeiligen Absätzen und Unterkapiteln.

<auto> – Method to use for autoconfiguration. If this is either 'rarp' or 'bootp' the specified protocol is being used. If the value is 'both' or empty, both protocols are used so far as they have been enabled during kernel configuration Using 'none' means no autoconfiguration. In this case you have to specify all necessary values in the fields before.

The <auto> parameter can appear alone as the value to the nfsaddrs parameter (without all the ':' characters before) in which case autoconfiguration is used. However, the 'none' value is not available in that case.

3.5 Other Misc. Kernel Boot Arguments

These various boot arguments let the user tune certain internal kernel parameters.

3.5.1 The 'debug' Argument

The kernel communicates important (and not-so important) messages to the operator via the `printk()` function. If the message is considered important, the `printk()` function will put a copy on the present console as well as handing it off to the `klogd()` facility so that it gets logged to disk. The reason for printing important messages to the console as well as logging them to disk is because under unfortunate circumstances (e.g. a disk failure) the message won't make it to disk and will be lost.

The threshold for what is and what isn't considered important is set by the `console_loglevel` variable. The default is to log anything more important than `DEBUG` (level 7) to the console. (These levels are defined in the include file `kernel.h`) Specifying `debug` as a boot argument will set the console loglevel to 10, so that *all* kernel messages appear on the console.

The console loglevel can usually also be set at run time via an option to the `klogd()` program. Check the man page for the version installed on your system to see how to do this.

3.5.2 The 'init=' Argument

The kernel defaults to starting the 'init' program at boot, which then takes care of setting up the computer for users via launching getty programs, running 'rc' scripts and the like. The kernel first looks for `/sbin/init`, then `/etc/init` (depreciated), and as a last resort, it will try to use `/bin/sh` (possibly on `/etc/rc`). If for example, your init program got corrupted and thus stopped you from being able to boot, you could simply use the boot prompt `init=/bin/sh` which would drop you directly into a shell at boot, allowing you to replace the corrupted program.

3.5.3 The 'no387' Argument

Some i387 coprocessor chips have bugs that show up when used in 32 bit protected mode. For example, some of the early ULSI-387 chips would cause solid lockups while performing floating point calculations, apparently due to a bug in the FRSAV/FRRESTOR instructions. Using the 'no387' boot argument causes Linux to ignore the math coprocessor even if you have one. Of course you must then have your kernel compiled with math emulation support! This may also be useful if you have one of those *really* old 386 machines that could use an 80287 FPU, as linux can't use an 80287.

3.5.4 The 'no-hlt' Argument

The i386 (and successors thereof) family of CPUs have a 'hlt' instruction which tells the CPU that nothing is going to happen until an external device (keyboard, modem, disk, etc.) calls upon the CPU to do a task. This allows the CPU to enter a 'low-power' mode where it sits like a zombie until an external device wakes it

Abbildung A.3: Beispiel einer Seite mit Unter-Unterkapiteln und normalen Absätzen.

5 Interpreting the Basic Specifications

This section explains what the specifications above mean, and some other things you'll need to know. First, some definitions. Next to each in parens is the variable name we'll use for it when doing calculations

horizontal sync frequency (HSF)

Horizontal scans per second (see above).

vertical sync frequency (VSF)

Vertical scans per second (see above). Mainly important as the upper limit on your refresh rate.

dot clock (DCF)

More formally, 'driving clock frequency'; The frequency of the crystal or VCO on your adaptor — the maximum dots-per-second it can emit.

video bandwidth (VB)

The highest frequency you can feed into your monitor's video input and still expect to see anything discernible. If your adaptor produces an alternating on/off pattern, its lowest frequency is half the DCF, so in theory bandwidth starts making sense at DCF/2. For tolerately crisp display of fine details in the video image, however, you don't want it much below your highest DCF, and preferably higher.

frame length (HFL, VFL)

Horizontal frame length (HFL) is the number of dot-clock ticks needed for your monitor's electron gun to scan one horizontal line, *including the inactive left and right borders*. Vertical frame length (VFL) is the number of scan lines in the *entire* image, including the inactive top and bottom borders.

screen refresh rate (RR)

The number of times per second your screen is repainted (this is also called "frame rate"). Higher frequencies are better, as they reduce flicker. 60Hz is good, VESA-standard 72Hz is better. Compute it as

```
RR = DCF / (HFL * VFL)
```

Note that the product in the denominator is *not* the same as the monitor's visible resolution, but typically somewhat larger. We'll get to the details of this below.

The rates for which interlaced modes are usually specified (like 87Hz interlaced) are actually the half-frame rates: an entire screen seems to have about that flicker frequency for typical displays, but every single line is refreshed only half as often.

For calculation purposes we reckon an interlaced display at its full-frame (refresh) rate, i.e. 43.5Hz. The quality of an interlaced mode is better than that of a non-interlaced mode with the same full-frame rate, but definitely worse then the non-interlaced one corresponding to the half-frame rate.

5.1 About Bandwidth:

Monitor makers like to advertise high bandwidth because it constrains the sharpness of intensity and color changes on the screen. A high bandwidth means smaller visible details.

Your monitor uses electronic signals to present an image to your eyes. Such signals always come in in wave form once they are converted into analog form from digitized form. They can be considered as combinations of many simpler wave forms each one of which has a fixed frequency, many of them are in the Mhz range, eg, 20Mhz, 40Mhz, or even 70Mhz. Your monitor video bandwidth is, effectively, the highest-frequency analog signal it can handle without distortion.

Abbildung A.4: Beispiel einer Seite mit Beschreibungen, die teilweise aus mehreren Absätzen bestehen und mit vorformatiertem Text gemischt sind.

```
rescue:~# dd if=/mnt/MBR of=/dev/hda bs=446 count=1
```

assuming that a floppy containing `MBR` is mounted under `/mnt`. Alternatively, use a DOS rescue floppy to issue `FDISK /MBR`.

2.12 Printer Configuration

Red Hat and Caldera have a fine configuration tool, `printtool`; if you don't use these distributions, manual configuration follows.

Let's suppose you have a non-PostScript printer you want to use to print raw text (e.g., C source files) and PostScript files via Ghostscript, which is assumed to be already installed.

Setting up the printer involves a few steps:

- find out which one the parallel print device is: try

  ```
  ~# echo "hello, world" > /dev/lp0
  ~# echo "hello, world" > /dev/lp1
  ```

  and take note which one works.

- make two spool directories:

  ```
  ~# cd /var/spool/lpd
  /var/spool/lpd/# mkdir raw ; mkdir postscript
  ```

- if your printer exibits the "staircase effect" (most inkjets do), you'll need a filter. Try to print two lines with

  ```
  ~# echo "first line" > /dev/lp1 ; echo "second line" > /dev/lp1
  ```

  if the output is like this:

  ```
  first line
          second line
  ```

  then save this script as `/var/spool/lpd/raw/filter`:

  ```
  #!/bin/sh
  # This filter does away with the "staircase effect"
  awk '{print $0, "\r"}'
  ```

  and make it executable with `chmod 755 /var/spool/lpd/raw/filter`.

- make a filter for PostScript emulation. Write the following filter as `/var/spool/lpd/postscript/filter`:

  ```
  #!/bin/sh

  DEVICE=djet500
  RESOLUTION=300x300
  PAPERSIZE=a4
  SENDEOF=

  nenscript -TUS -ZB -p- |
  if [ "$DEVICE" = "PostScript" ]; then
  ```

Abbildung A.5: Beispiel einer Seite mit Auflistungen, die teilweise aus mehreren Absätzen bestehen und mit vorformatiertem Text gemischt sind.

8 FAQs

This is the FAQ section. Most of it was written by Alan Cox.

1. I get a lot of 'stale nfs handle' errors when using Linux as a nfs server.

   This is caused by a bug in some oldish nfsd versions. It is fixed in nfs-server2.2beta16 and later.

2. When I try to mount a file system I get

   ```
   can't register with portmap: system error on send
   ```

   You are probably using a Caldera system. There is a bug in the rc scripts. Please contact Caldera to obtain a fix.

3. Why can't I execute a file after copying it to the NFS server?

   The reason is that nfsd caches open file handles for performance reasons (remember, it runs in user space). While nfsd has a file open (as is the case after writing to it), the kernel won't allow you to execute it. Nfsds newer than spring 95 release open files after a few seconds, older ones would cling to them for days.

4. My NFS files are all read only

   The Linux NFS server defaults to read only. RTFM the "exports" and nfsd manual pages. You will need to alter `/etc/exports`.

5. I mount from a linux nfs server and while ls works I can't read or write files.

   On older versions of Linux you must mount a NFS servers with `rsize=1024,wsize=1024`.

6. I mount from a Linux NFS server with a block size of between 3500-4000 and it crashes the Linux box regularly

   Basically don't do it then.

7. Can Linux do NFS over TCP

   No, not at present.

8. I get loads of strange errors trying to mount a machine from a Linux box.

   Make sure your users are in 8 groups or less. Older servers require this.

9. When I reboot my machine it sometimes hangs when trying to unmount a hung NFS server.

   Do **not** unmount NFS servers when rebooting or halting, just ignore them, it will not hurt anything if you don't unmount them. The command is `umount -avt nonfs`.

10. Linux NFS clients are very slow when writing to Sun and BSD systems

    NFS writes are normally synchronous (you can disable this if you don't mind risking losing data). Worse still BSD derived kernels tend to be unable to work in small blocks. Thus when you write 4K of data from a Linux box in the 1K packets it uses BSD does this

    ```
    read 4K page
    alter 1K
    write 4K back to physical disk
    read 4K page
    alter 1K
    write 4K page back to physical disk
    etc..
    ```

Abbildung A.6: Beispiel einer Seite mit Aufzählungen, die teilweise aus mehreren Absätzen bestehen und mit vorformatiertem Text gemischt sind.

2.4 Hard Disk Performance

Your hard disk's performance can be greatly enhanced by *carefully* using hdparm(8). If your Linux distribution doesn't include it, you'll find on <ftp://sunsite.unc.edu/pub/Linux/system/hardware> ; look for a file called hdparm-X.Y.tar.gz.

I can't give you a general recipe, as many details depend on your hard disk and HD controller. Since you risk to toast your filesystem, please *read the man page carefully* before using some options. At its simplest, you could add the following line to /etc/rc.d/rc.sysinit:

```
/sbin/hdparm -c1 /dev/hda  # first IDE drive assumed
```

which enables (E)IDE 32-bit I/O support. As for the '-m' option, this is what hdparm author Mark Lord emailed me:

> (...) if your system uses components from the past couple of years, it will be fine. Older than that, there *may* be a problem (unlikely). The really buggy chips were the CMD0646 and RZ1000 chips, used *extensively* on 486 and (early) 586 motherboards about 2-3 years ago.

2.5 Parallel Port Zip Drive

To use the parallel port version of the Zip drive you can use the default driver that comes with recent (2.x.x) kernels. During kernel configuration, make sure that SCSI support and SCSI disk support are enabled (either in the kernel or as a module). Remember, there can be conflicts between the printer and the Zip drive on the same parallel port.

Zip disks are sold preformatted on partition /dev/sda4. To enable the Zip, all you have to do is issue

```
#~ chmod 666 /dev/sda4  # everyone can access the Zip Drive
#~ insmod ppa
```

and the Zip can now be mounted as usual (better write the last line in /etc/rc.d/rc.sysinit). You also access the Zip drive via mtools adding this line to your /etc/mtools.conf:

```
drive z: file="/dev/sda4" exclusive
```

There's a better ppa driver than the standard one, though: have a look at <http://www.torque.net/~campbell> .

2.6 Device Drivers

Devices in /dev (or better, links to the actual device drivers) may be missing. Check what devices your mouse, modem, and CD-ROM drive correspond to, then do what follows:

```
~# cd /dev
/dev# ln -s /dev/cua0 mouse
/dev# ln -s /dev/cua1 modem
/dev# ln -s /dev/hdb cdrom
```

and, if you want, do chmod 666 to these devices (not the links, the actual devices!) to make them fully accessible by every user. Tip: in some laptops the mouse device is /dev/psaux: take this into account when configuring X11.

In addition, you'll want to make the floppy accessible by non-root users with chmod 666 /dev/fd*. This is bound to cause security problems, but I don't know the details. Comments are welcome.

Abbildung A.7: Beispiel einer Seite mit Zitat.