Master-Arbeit

IceCite: Ein System zur Verwaltung von Wissenschaftlichen Publikationen mit Automatischer Metadaten-Extraktion

Claudius Korzen

Gutachterin:
Prof. Dr. Hannah Bast

Betreuerin:
Prof. Dr. Hannah Bast

Institut für Informatik
Lehrstuhl für Algorithmen und Datenstrukturen
Albert-Ludwigs-Universität Freiburg

Erklärung

Hiermit erkläre ich, dass ich diese Masterarbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Masterarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, 11. Oktober 2011

Claudius Korzen

Kurzfassung

Wir präsentieren in dieser Masterarbeit IceCite – ein System, mit dem wissenschaftliche Publikationen verwaltet und verwandte Publikationen gefunden werden können. Das System bestimmt die Metadaten und bibliografischen Daten (Literatur-Referenzen) von Publikationen automatisch, indem es Titel und Referenzen aus entsprechenden PDF-Dateien extrahiert und Einträgen aus den Literaturdatenbanken *DBLP* und *Medline* zuordnet. Wir erhalten damit vollständige und korrekte Metadaten, die sowohl die Publikationen als auch ihre Referenzen eindeutig identifizieren. Unter anderem weil wir damit in der Lage sind, Verknüpfungen zwischen Publikationen herzustellen, kann das System alle notwendigen Schritte einer Literaturrecherche vereinfachen.

Die Schwierigkeit der Titel- und Referenzen-Extraktion aus wissenschaftlichen Publikationen liegt in der großen Variabilität in Position und Struktur der zu extrahierenden Elemente. Mögliche Fehler in den extrahierten Zeichenketten, die z.B. auf Schreib- oder Extraktionsfehler zurückzuführen sind, machen zudem eine *fehlertolerante* Zuordnung zu einem Literaturdatenbank-Eintrag notwendig.

In ausführlichen Experimenten an 700 Publikationen aus *DBLP* und 500 Publikationen aus *Medline* haben wir unter anderem die Korrektheit der Ergebnisse unserer Algorithmen zur Extraktion und Zuordnung von Titeln und Referenzen evaluiert. Wir erreichen Ergebnisse von bis zu 98.8% korrekt extrahierter Titel und von bis zu 91.1% korrekt extrahierter Referenzen. Die Zuordnung zu repräsentierenden Einträgen aus DBLP und Medline gelingt uns für bis zu 99.9% Titel und für bis zu 94.8% Referenzen.

Inhaltsverzeichnis

Kapitel 1

Einführung

Angenommen, Sie sind als Wissenschaftler mit einem Forschungsthema beschäftigt, für das Sie gerade eine umfangreiche Literaturrecherche durchführen, um sich über den aktuellen Forschungsstand dieses Themas zu informieren. Üblicherweise gehen Sie dabei in den folgenden drei Schritten vor[1]:

1. **Literatursuche:** Zuerst ist eine Suche nach Literatur zu Ihrem Thema notwendig. Hierfür eignen sich spezielle Literaturdatenbanken (z.B. DBLP oder Medline, s.u.) und Literatursuchmaschinen. Möglich ist auch eine sogenannte *Schneeballsuche*, d.h. die Suche nach Publikationen, die in Ihnen schon bekannten Publikationen referenziert werden. Da die referenzierten Publikationen oft ein ähnliches Thema behandeln, handelt es sich bei diesen Publikationen meist ebenfalls um relevante Literatur. Die Schneeballsuche können Sie beliebig oft wiederholen, indem Sie in den referenzierten Publikationen wiederum nach referenzierten Publikationen suchen, usw.

2. **Literaturauswahl:** Für jede der aus der Literatursuche erhaltenen Publikationen müssen Sie nun entscheiden, welche relevant für Sie sind und detaillierter studieren wollen.

3. **Literaturbeschaffung:** Um die ausgewählten Publikationen lesen und gegebenenfalls auf Ihrem Rechner speichern zu können, müssen Sie nun die entsprechenden Dateien (meistens verfügbar als PDF-Dateien) beschaffen. Oft bieten Literaturdatenbanken bzw. Literatursuchmaschinen dazu entsprechende Download-Links an. Da aus Gründen des Urheberrechts viele Publikationen nur Mitgliedern wissenschaftlicher Einrichtungen zugänglich sind, muss ein Download meistens aus dem Rechnernetz solch einer Einrichtung erfolgen.

Je nach Vorgehensweise und nach Menge der verfügbaren Literatur können sich alle drei Schritte als sehr mühsam und zeitaufwändig herausstellen. So müssen Sie z.B. für die oben erwähnte Schneeballsuche zunächst die PDF-Datei der Publikation öffnen, um anschließend das Literaturverzeichnis ausfindig zu machen. Eine alternative Methode ist die Verwendung von Literatursuchmaschinen wie z.B. *CiteSeer$^{\chi}$* [4] oder *Google Scholar* [10], die auch die Referenzen wissenschaftlicher Publikationen indizieren und durchsuchbar machen. Diese Funktion beschränkt sich allerdings auf frei verfügbare Publikationen, so dass mit dieser Methode möglicherweise nur eine unvollständige Schneeballsuche möglich ist.

Zudem können Sie aufgrund des beschränkten Informationsgehaltes eines Referenz-Eintrages nicht immer auf Anhieb entscheiden, ob die referenzierte Publikation tatsächlich relevant für Sie ist. Deshalb geht mit der Literaturauswahl fast immer auch die Literaturbeschaffung einher, um anhand des Inhaltes der Publikation die Relevanz für ihr Forschungsthema zu bestimmen.

Wollen Sie die beschaffenen Publikationen für eine spätere Bearbeitung auf ihrem Rechner speichern, sehen Sie sich außerdem mit der Herausforderung konfrontiert, geeignete Namen

[1] Es wird die Literaturrecherche in digitalen Datenbeständen beschrieben. Die Literaturrecherche in Printmedien wird an dieser Stelle nicht berücksichtigt.

für die Dateien zu vergeben, da die von den Verlegern vorgegebenen Dateinamen nicht besonders aussagekräftig sind: Als Beispiele seien hier *Springer* [22], dessen PDF-Dateien grundsätzlich `fulltext.pdf` heißen, *IEEE* [13], dessen PDF-Dateien numerische ID's im Dateinamen (z.B. `05763396.pdf`) enthalten, oder *ACM* [1], dessen Dateinamen zumindest den Namen des ersten Autoren der Publikation enthalten (z.B. `p48-mccallum.pdf`), genannt. Für das schnelle Identifizieren einer bestimmten Publikation sind aussagekräftige Dateinamen zwar unumgänglich, die Umbenennung der Dateien benötigt aber zusätzlichen Aufwand. Machen Sie sich hingegen diese Mühe nicht, so ist – je nach Menge der gespeicherten Publikationen – die Verwechslungsgefahr groß und das Finden einer bestimmten Publikation mindestens genauso aufwändig.

Motiviert durch die beschriebenen Schwierigkeiten bei einer Literaturrecherche ist die Idee zu dieser Masterarbeit entstanden. Wir präsentieren ein System, mit dem es möglich ist, seine gesammelten wissenschaftlichen Publikationen in einer persönlichen Bibliothek zu verwalten. Das System bestimmt automatisch sowohl die Metadaten (z.B. Titel, Autoren, Erscheinungsjahr, etc.) einer Publikation als auch die Metadaten der referenzierten Publikationen, um jede Publikation eindeutig zu beschreiben und identifizierbar zu machen. Durch die Identifizierung der Referenzen einer Publikation wird es unter anderem möglich sein, verwandte Publikationen zu finden und per Knopfdruck der Bibliothek automatisch hinzuzufügen. Eine Schneeballsuche ist damit sehr schnell und komfortabel durchführbar. Zudem sind die Metadaten und die Volltexte aller Publikationen durchsuchbar.

Das System wurde von uns vor allem mit der Absicht implementiert, dem Benutzer bei einer Literaturrecherche so viel Arbeit wie möglich abzunehmen und dabei eine möglichst intuitive Bedienbarkeit des Systems zu ermöglichen. Dies impliziert auch möglichst niedrige Antwortzeiten des Systems. Alle nun folgenden Ziele dieser Masterarbeit werden wir deshalb besonders unter Berücksichtigung des Aspekts *Effizienz* verfolgen.

1.1 Ziele

Um die korrekten Metadaten zu einer wissenschaftlichen Publikation zu erhalten, extrahieren wir aus einer entsprechenden PDF-Datei zunächst ihren Titel (dieser Vorgang sei im weiteren Verlauf *Titel-Extraktion* genannt). Im Anschluss versuchen wir, die Publikation mithilfe dieses Titels ihrem repräsentierenden Eintrag aus einer Literaturdatenbank zuzuordnen (*Titel-Zuordnung*), sofern solch ein Eintrag überhaupt existiert. Eine Literaturdatenbank ist eine Datenbank, die eine Menge von Einträgen enthält, in denen bibliografische Angaben zu wissenschaftlichen Publikationen – üblicherweise getrennt in definierte Metadaten-Felder, wie z.B. Titel, Autoren, Erscheinungsjahr, Konferenz und einem eindeutigen Key – gespeichert sind. In Beispiel 1.1 ist ein Eintrag aus der Literaturdatenbank *DBLP*[2] zu sehen. *DBLP* ist eine Literaturdatenbank aus dem Bereich der Informatik, die insgesamt mehr als 1,7 Millionen Einträge ([42], Stand: September 2011) in einer XML-Datei `dblp.xml` speichert. Der abgebildete Eintrag enthält neben dem Datum, an dem der Eintrag zuletzt geändert wurde (`mdate`) und einem eindeutigen Identifizierer (`key`), die Autoren (`author`), den Titel (`title`) und das Erscheinungsjahr (`year`) der Publikation, sowie den Titel des Konferenzbandes (`booktitle`) und die Position, an der die Publikation innerhalb des Konferenzbandes zu finden ist (`pages`). Zusätzlich enthält der Eintrag eine URL zu der Seite, auf der der Verleger die Publikation als digitales Dokument verbreitet (`ee`; steht für electronic edition) und eine URL zu einer Webseite auf den DBLP-Servern, auf der weitere Informationen zur Publikation zu finden sind (z.B. ein vorgefertigter BibTeX-Eintrag der Publikation). Alle Details zu DBLP werden wir in Kapitel 4.1 ebenso besprechen, wie zu der Literaturdatenbank *Medline* aus dem Bereich der Medizin mit knapp 19 Millionen Einträgen ([23], Stand: August 2011).

Ähnlich gehen wir bei den Referenzen vor: Zunächst extrahieren wir jede Referenz aus dem Literaturverzeichnis einer Publikation (*Referenzen-Extraktion*), um sie danach mithilfe dieses Extrakts jeweils ihrem repräsentierenden Eintrag in einer Literaturdatenbank zuzuordnen (*Referenzen-Zuordnung*).

[2] DBLP konzentrierte sich ursprünglich auf die Gebiete *DataBase Systems* und *Logic Programming*. Heutzutage deckt DBLP nahezu alle Gebiete der Informatik ab, weshalb die Abkürzung als *Digital Bibliography & Library Project* gelesen werden kann [7].

```
<dblp>
   [...]
   <inproceedings mdate="2002-12-09" key="conf/sigcomm/KrishnamurthyW00">
      <author>Balachander Krishnamurthy</author>
      <author>Jia Wang</author>
      <title>On network-aware clustering of web clients.</title>
      <pages>97-110</pages>
      <year>2000</year>
      <booktitle>SIGCOMM</booktitle>
      <ee>http://doi.acm.org/10.1145/347059.347412</ee>
      <url>db/conf/sigcomm/sigcomm2000.html#KrishnamurthyW00</url>
   </inproceedings>
   [...]
</dblp>
```

Beispiel 1.1: Ausschnitt aus der XML-Datei `dblp.xml` der Literaturdatenbank *DBLP*. Der abgebildete Eintrag enthält Attribute namens `mdate` und `key` sowie die Felder `author`, `title`, `pages`, `year`, `booktitle`, `ee` und `url`.

Das System setzt sich neben der *Titel-Extraktion* und der *Titel-Zuordnung* sowie der *Referenzen-Extraktion* und der *Referenzen-Zuordnung* zusätzlich aus den Teilkomponenten *Suche* und *Benutzerschnittstelle* zusammen. Jede Komponente verfolgt für sich ein individuelles Ziel, um eine Teilfunktionalität des Systems zu realisieren. Diese Ziele sind:

1. **Titel-Extraktion:** *Gegeben sei eine PDF-Datei einer wissenschaftlichen Publikation. Extrahiere den vollständigen und korrekten Titel der Publikation aus der PDF-Datei.*

   Um die vorliegende Publikation zu identifizieren und anschließend ihre Metadaten zu bestimmen, soll zunächst der Titel aus der entsprechenden PDF-Datei extrahiert werden. Dieser dient der nächsten Komponente als Eingabe, die folgendes Ziel hat:

2. **Titel-Zuordnung:** *Gegeben sei ein (extrahierter) Titel einer wissenschaftlichen Publikation sowie (mindestens) eine Literaturdatenbank. Finde für den extrahierten Titel den repräsentierenden Eintrag in der Literaturdatenbank.*

   Das Ziel ist, die Publikation, aus der der Titel extrahiert wurde, ihrem repräsentierenden Eintrag einer Literaturdatenbank zuzuordnen. Diesem entnehmen wir die korrekten und vollständigen Metadaten, die wir mit der Publikation assoziieren.

3. **Referenzen-Extraktion:** *Gegeben sei eine PDF-Datei einer wissenschaftlichen Publikation. Identifiziere das Literaturverzeichnis in der PDF-Datei und extrahiere daraus jede enthaltene Referenz.*

   Um die zitierten Publikationen genauso zu identifizieren, sollen alle im Literaturverzeichnis enthaltenen Referenzen extrahiert werden. Um auch zu jeder Referenz ihre Metadaten zu erhalten, leiten wir diese Extrakte an die nächste Komponente mit folgendem Ziel weiter:

4. **Referenzen-Zuordnung:** *Gegeben seien die (extrahierten) Referenzen einer wissenschaftlichen Publikation sowie (mindestens) eine Literaturdatenbank. Finde für jede Referenz ihren repräsentierenden Eintrag in der Literaturdatenbank.*

   Analog zur Titel-Zuordnung sollen hier die Referenzen einer Publikation einem Eintrag einer Literaturdatenbank zugeordnet werden. Die dadurch erhaltenen Metadaten assoziieren wir mit der jeweiligen Referenz.

5. **Suche:** *Gegeben sei eine Bibliothek mit wissenschaftlichen Publikationen, eine Menge von Literaturdatenbanken, und eine Suchanfrage q. Durchsuche die Metadaten und Volltexte aller Publikationen aus der Bibliothek sowie alle Einträge aus den Literaturdatenbanken und gebe diejenigen Publikationen und Einträge aus, die relevant für q sind.*

   Mit der Suche soll vor allem ein schnelles Auffinden von zu einer Suchanfrage relevanten Informationen, z.B. Publikationen zu einem bestimmten Thema, ermöglicht werden.

6. **Bereitstellen einer Benutzerschnittstelle:** *Gegeben sei das System, das sich aus oben genannten Komponenten zusammensetzt. Stelle eine Benutzerschnittstelle zur Verfügung, die das System für Benutzer zugänglich macht und die Verwaltung von wissenschaftlichen Publikationen sowie das einfache Finden verwandter Publikationen ermöglicht.*

   Mit der Benutzerschnittstelle soll dem Benutzer die Möglichkeit gegeben werden, wissenschaftliche Publikation in einer persönlichen Bibliothek zu sammeln, zu verwalten, durchsuchbar zu machen und verwandte Publikationen zu finden.

1.2 Systemarchitektur

Der zentrale Punkt für einen Benutzer des Systems ist die Benutzerschnittstelle, über die er das System nutzen und wissenschaftliche Publikationen verwalten kann. Sobald er PDF-Dateien von Publikationen zu seiner Bibliothek hinzufügt, werden zunächst die Titel und Referenzen extrahiert und jeweils zu Einträgen einer Literaturdatenbank zugeordnet. Die Extraktion und Zuordnung der Titel sind dabei genauso eng verzahnt wie die Extraktion und Zuordnung der Referenzen und werden in der Regel jeweils zusammen ausgeführt. Aus den zugeordneten Literaturdatenbank-Einträgen entnehmen wir die Metadaten, die wir dazu nutzen, die Publikationen in der Bibliothek zu beschreiben. Diese Metadaten sind zusammen mit den Volltexten der Publikationen und allen Einträgen aus den Literaturdatenbank(en) durchsuchbar, um zu einer Suchanfrage relevante Publikationen zu finden.

In Abbildung 1.2 ist die gerade beschriebene Architektur schematisch dargestellt. Die Kernkomponenten sind blau hervorgehoben und die üblichen Ablaufrichtungen durch Pfeile gekennzeichnet. Die Literaturdatenbank(en) und die Bibliothek eines Benutzers – in der Abbildung grau dargestellt – bilden als entscheidende Informationslieferanten das Rückgrat des Systems und stehen im ständigen Austausch mit der Suche-Komponente und den Zuordnungs-Komponenten.

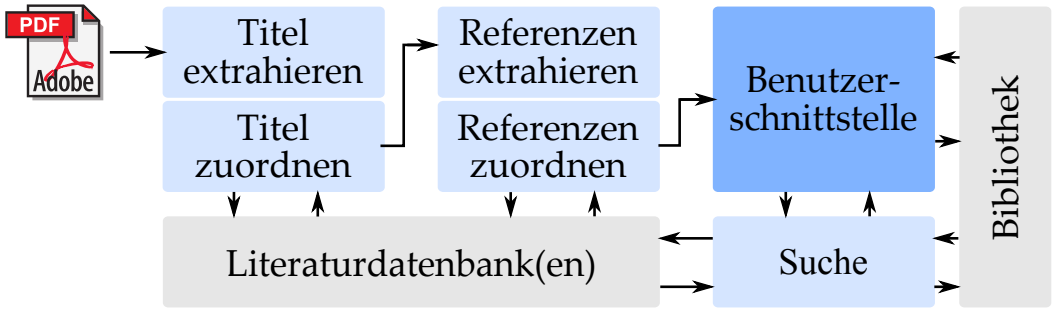


Abbildung 1.2: Das Zusammenspiel der Komponenten des Systems im Überblick. Bevor eine Publikation der Bibliothek hinzugefügt wird, wird der Titel extrahiert und einem Eintrag einer Literaturdatenbank zugeordnet. Danach werden die Referenzen extrahiert, die jeweils auch einem Eintrag einer Literaturdatenbank zugeordnet werden. Mit der Benutzerschnittstelle können alle Publikationen der Bibliothek mit den ermittelten Metadaten verwaltet werden. Für eine Suche sind sowohl die Literaturdatenbanken als auch die persönliche Bibliothek durchsuchbar.

Um die Grundidee des gesamten Systems zu veranschaulichen, sei im Folgenden kurz der wesentliche Aufbau der Benutzerschnittstelle erläutert: Die Benutzerschnittstelle ist hauptsächlich in drei Sichten aufgeteilt, nämlich in die *Login-Sicht*, in die *Bibliothekssicht* und die *Detailsicht*. Während die Login-Sicht lediglich dazu dient, dass sich ein Benutzer anmelden kann, beinhalten die Bibliothekssicht und die Detailsicht die Hauptfunktionalitäten der Benutzerschnittstelle.

Bibliothekssicht

Die Bibliothekssicht ist unterteilt in eine Kopfzeile (vgl. Bereich 1 in Abbildung 1.3), in eine Werkzeugleiste (Bereich 2), in eine Publikationsliste (Bereich 3), in eine Referenzenliste (Be-

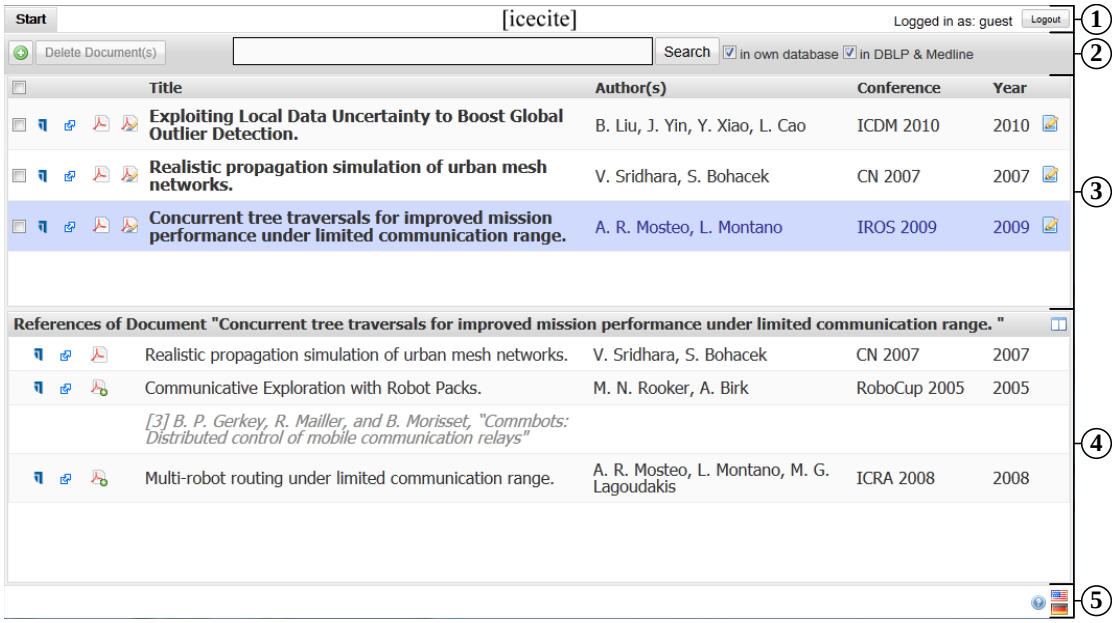


Abbildung 1.3: Die Bibliothekssicht der Anwendung mit Kopfzeile (Bereich 1), Werkzeugleiste (Bereich 2), Publikationsliste (Bereich 3), Referenzenliste (Bereich 4) und Fußzeile (Bereich 5).

reich 4) und in eine Fußzeile (Bereich 5). In der Publikationsliste werden alle Publikationen der persönlichen Bibliothek mit automatisch bestimmten Metadaten aufgelistet. Nach Klick auf eines der PDF-Symbole (oder) wird die *Detailsicht* einer Publikation eingeblendet. Die Referenzenliste zeigt alle Referenzen einer ausgewählten Publikation an. Wenn eine extrahierte Referenz einem Eintrag in einer Literaturdatenbank zugeordnet werden konnte, wird sie ebenfalls mit ihren automatisch bestimmten Metadaten aufgelistet. Anderenfalls wird die extrahierte Zeichenkette der Referenz mit einer grauen Schriftart angezeigt (wie es für die dritte Referenz in der Referenzenliste aus Abbildung 1.3 der Fall ist). Falls eine Referenz bereits als Publikation in der Bibliothek hinterlegt ist, wird für sie das PDF-Symbol mit einem Link zur Detailsicht der Publikation angezeigt. Ansonsten wird für eine Referenz das Symbol angezeigt. Nach Klick auf dieses wird im Internet automatisch nach einer PDF-Datei der entsprechenden Publikation gesucht, aus der zunächst die benötigten Metadaten bestimmt werden und dann automatisch der Bibliothek hinzugefügt wird.

Durch die Eingabe einer Suchanfrage in das Textfeld der Werkzeugleiste können zudem sowohl die Publikationen aus der persönlichen Bibliothek inklusive ihrer Volltexte als auch die Literaturdatenbanken DBLP und Medline durchsucht werden.

Detailsicht

Nach Klick auf eines der PDF-Symbole einer Publikation in der Bibliothekssicht wird die Detailsicht einer Publikation eingeblendet. Die Detailsicht zeigt zusammen mit den Metadaten (vgl. Bereich 1 in Abbildung 1.4) und den Referenzen einer Publikation (Bereich 2) die entsprechende PDF-Datei der Publikation an (Bereich 3). Bei der PDF-Datei handelt es sich entweder um die originale PDF-Datei (nach Klick auf in der Bibliothekssicht) oder um eine annotierte Version (nach Klick auf), die während der Extraktion der Referenzen erstellt wurde. In einer annotierten Version erhält jede Referenz eine Annotation, in der z.B. eine individuelle URL hinterlegt werden kann, die auf eine Webseite mit weiterführenden Informationen zur referenzierten Publikation verweist. Ausführliche Details sowohl zu annotierten Versionen von PDF-Dateien als auch zu den restlichen Funktionalitäten der Benutzerschnittstelle sind in Kapitel 6 zu finden.

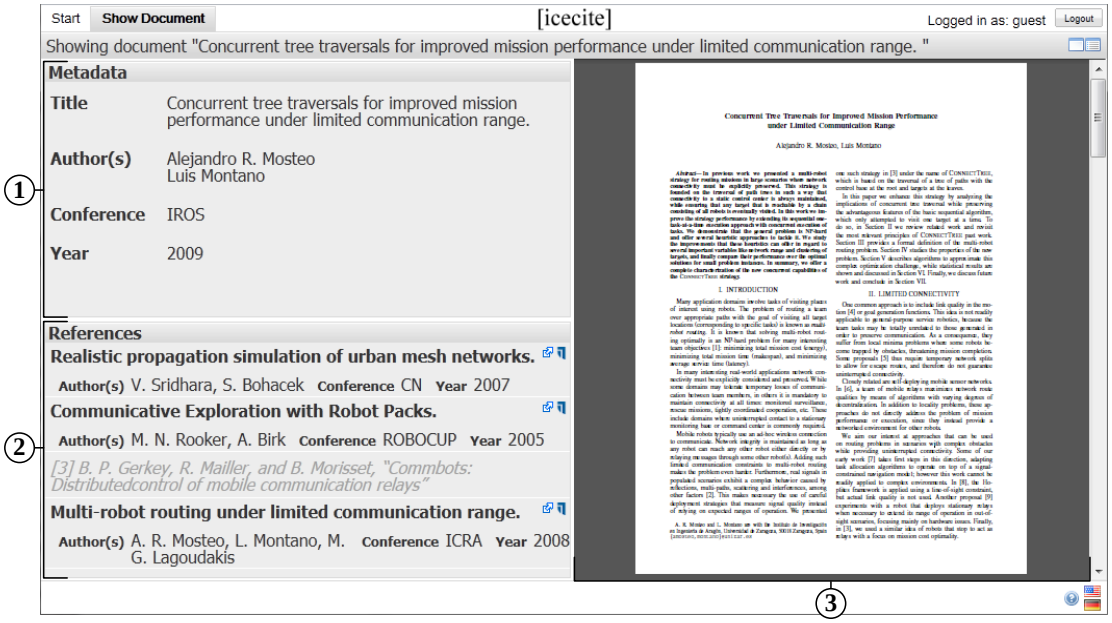


Abbildung 1.4: Die Detailsicht einer Publikation mit Metadaten (Bereich 1), Referenzenliste (Bereich 2) und PDF-Betrachter (Bereich 3).

1.3 Aufbau der Arbeit

Wir haben nun einen Überblick über die Ziele und den Umfang dieser Arbeit und können, nachdem wir in Kapitel 2 verwandte Literatur zu den in dieser Masterarbeit behandelten Themen betrachtet haben, damit beginnen, die Themen detailliert zu diskutieren. In Kapitel 3 werden wir die Extraktion von Titeln und Referenzen aus PDF-Dateien wissenschaftlicher Publikationen diskutieren. Die anschließende Zuordnung der extrahierten Daten zu Einträgen aus Literaturdatenbanken wird Thema des Kapitels 4 sein. Um die Qualität der Ergebnisse unserer Algorithmen bewerten zu können, werden wir in Kapitel 5 einige Experimente durchführen, mit denen wir neben der Korrektheit der Ergebnisse auch die Effizienz unserer Algorithmen untersuchen. Schließlich werden wir in Kapitel 6 alle Funktionalitäten der Benutzerschnittstelle vorstellen und erläutern, inwiefern sie alle Schritte einer Literaturrecherche und die Suche nach verwandten Publikationen vereinfachen können.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel werden wir Arbeiten vorstellen, die zu den Themen, die in dieser Masterarbeit diskutiert werden, verwandt sind. Dazu werden wir zunächst das Thema der Extraktion von Metadaten aus wissenschaftlichen Publikationen betrachten und danach mögliche Ansätze zur Zuordnung von extrahierten Zeichenketten zu repräsentierenden Einträgen in Literaturdatenbanken diskutieren. Schließlich stellen wir verwandte Software vor, die zum Zweck der Literaturrecherche implementiert wurden.

2.1 Methoden der Metadaten-Extraktion

Die Autoren von zur Titel- und Referenzen-Extraktion verwandten Arbeiten haben meist die Absicht, nicht nur den Titel bzw. die Referenzen aus wissenschaftlichen Publikationen zu extrahieren, sondern auch sämtliche verfügbaren Metadaten-Felder von Publikationen und Referenzen zu identifizieren. Oft werden hierzu Methoden des Maschinellen Lernens, seltener auch regelbasierte Heuristiken verwendet.

2.1.1 Methoden des Maschinellen Lernens

Unter Maschinellem Lernen versteht man den Ansatz, künstlichen Modellen durch Beispiele, für die die zu berechnenden Ergebnisse bereits bekannt sind, ein gewünschtes Verhalten beizubringen. Dadurch soll das Modell lernen, das Verhalten auf unbekannte Daten anzuwenden.

Für die Extraktion von Metadaten-Feldern bedeutet das, dass einem Modell beigebracht werden soll, innerhalb unbekannter Volltexte die Metadaten-Felder von Publikationen und Referenzen zu identifizieren. Dazu werden Modelle mit Daten, für die die Metadaten-Felder bereits bekannt sind, angelernt. Die verbreitetsten Methoden zur Metadaten-Extraktion sind *Hidden Markov Modelle*, *Support Vector Machines* und *Conditional Random Fields*.

Hidden Markov Modelle

Ein Hidden Markov Modell (HMM) ist ein probabilistischer, endlicher Automat und ist definiert durch eine endliche Menge von Zuständen, durch ein endliches Alphabet von Ausgabesymbolen, durch die Zustandsübergangswahrscheinlichkeiten und durch die Wahrscheinlichkeitsverteilung, mit der ein Zustand der Startzustand ist. Die Übergänge zwischen den Zuständen gehen mit der Ausgabe von Symbolen aus dem gegebenen Alphabet entsprechend einer Wahrscheinlichkeitsverteilung einher. Im Allgemeinen ist nur die Sequenz dieser Ausgabesymbole bekannt, aus der die Zustandsfolge, die diese Sequenz erzeugt hat, abzuleiten ist – die Zustandsfolge ist also „versteckt“ (*hidden*). Wie genau ein HMM definiert sein muss, um eine möglichst hohe Genauigkeit bei der Erkennung des Titels und der Referenzen einer Publikation zu erreichen, ist Gegenstand der Diskussion in der Literatur.

Connan und Omlin [31] unterscheiden den Aufbau von Referenzen in verschiedene Referenz-Stile und bauen für jeden Referenzstil ein eigenes HMM. Die Menge der Zustände eines

HMM entspricht dabei der Menge der Metadaten-Felder einer Publikation bzw. einer Referenz (Titel, Autoren, Erscheinungsjahr, etc.) und jedem Feld wird genau ein Zustand zugeordnet. Aus insgesamt 1.212 Referenzen haben Connan und Omlin sogenannte *Trainingsdaten* zusammengestellt, aus denen sie die Abfolge der Metadaten-Felder in den unterschiedlichen Referenzstilen untersucht und daraus die Übergangswahrscheinlichkeiten für die Zustände eines HMM berechnet haben. Das HMM, für das die Wahrscheinlichkeit am größten ist, eine gegebene Referenz erzeugt zu haben, definiert den Stil der Referenz und somit die Abfolge der Metadaten-Felder. Anhand dieser Abfolge können der Referenz die Werte ihrer Felder entnommen werden. Laut Connan und Omlin erreicht diese Methode zwar eine Genauigkeit von 97% korrekt identifizierter Felder, allerdings wurde dieser Wert lediglich mit schon bekannten Referenzstilen erreicht. Hier liegt unserer Meinung nach auch die Schwäche dieses Ansatzes: Für jeden Referenzstil muss ein eigenes HMM entworfen und angelernt werden. Dies führt dazu, dass nur bekannte Referenzstile erkannt werden können, was wir angesichts des Fehlens allgemeingültiger Normen für die Struktur von Referenzen und der damit verbundenen hohen Anzahl möglicher Referenzstile für nicht zweckmäßig erachten.

Einen universelleren Ansatz verfolgen Borkar et al. [30], indem sie ein geschachteltes HMM verwenden, das ebenfalls pro Metadaten-Feld einen Zustand besitzt. Jeder Zustand besteht wiederum aus einem HMM, das die Struktur innerhalb dieses Feldes abbildet. Mithilfe des Viterbi-Algorithmus [51] bestimmen Borkar et al. schließlich denjenigen Pfad, für den die Wahrscheinlichkeit, eine gegebene Referenz erzeugt zu haben, am größten ist. Sie erreichen damit eine Genauigkeit von 87,3% korrekt identifizierter Felder.

McCallum et al. [44] verwenden für die Extraktion der Metadaten und der Referenzen zwei verschiedene HMMs, die ihre innere Struktur von Trainingsdaten lernen. Dies geschieht, indem zuerst sogenannte *maximal spezifische* Modelle für die Trainingsdaten erstellt werden, die danach mit unterschiedlichen Merging-Techniken vereinfacht werden. McCallum et al. klassifizieren die Wörter innerhalb des Kopfbereiches und der Referenzen einer Publikation mit diesen HMMs in Metadaten-Felder und berichten von einer Rate von 7,3% falsch zugeordneter Metadaten-Felder (sprich einer Erfolgsrate von 92,7%) für die Publikationen und einer Rate von 6,6% falsch zugeordneter Metadaten-Felder (sprich einer Erfolgsrate von 93,4%) der Referenzen.

Erik Hetzner [36], der unter anderem pro Metadaten-Feld zwei Zustände und pro Abgrenzung eines Metadaten-Feldes einen Zustand verwendet, erreicht eine Genauigkeit von 93,4% korrekt identifizierter Metadaten-Felder von Referenzen. Yin et al. [52], die sogenannte *Bigramm HMMs* über Wort-Bigramme erstellen, berichten von einer Genauigkeit von 90,1% korrekt identifizierter Metadaten-Felder.

Hidden Markov Modelle scheinen also im Allgemeinen eine geeignete Methode zu sein, um die Titel und die Referenzen von wissenschaftlichen Publikationen zuverlässig extrahieren zu können.

Support Vector Machines

Ein weiterer Ansatz des Maschinellen Lernens sind sogenannte *Support Vector Machines* (SVMs), mit denen Daten (z.B. die Volltexte von Publikationen) klassifiziert werden können. Dazu werden die Daten auf Vektoren in einem Merkmalsraum abgebildet und durch eine optimal trennende Hyperebene in Klassen (z.B. Metadaten-Felder) eingeteilt. Die Schwierigkeit besteht darin, die Hyperebene, die die Volltexte in korrekte Metadaten-Felder klassifiziert, zu berechnen. Han et al. [34] benutzen SVMs, um die Metadaten einer wissenschaftlichen Publikation korrekt zu identifizieren. Dies gelingt ihnen mit einer Genauigkeit von 92,9%. Okada et al. [46] kombinieren SVMs mit HMMs und können damit zusätzlich die Metadaten-Felder von Referenzen einer Publikation identifizieren. Sie klassifizieren dazu die Felder der Referenzen mit SVMs. Ist diese Klassifizierung nicht eindeutig, so setzen sie zusätzlich HMMs ein, um die Abfolge der Felder bestimmen zu können. Okada et al. erreichen damit eine Genauigkeit von 98,8% korrekt identifizierter Metadaten-Felder von Referenzen.

Conditional Random Fields

Auch *Conditional Random Fields* (CRFs) sind in das Gebiet des Maschinellen Lernens einzuordnen und können ebenfalls zum Zweck der Titel- und Referenzen-Extraktion eingesetzt werden. CRFs bestehen aus einer ungerichteten, graphischen Struktur und können u.a. bedingte Wahrscheinlichkeitsverteilungen für Zeichenketten berechnen. Ziel ist es, die Metadaten-Felder innerhalb des Volltextes einer Publikation zu markieren, um anschließend die Werte für die Felder identifizieren zu können. Für eine Eingabesequenz $X = x_1, \ldots, x_T$ (z.B. eine geeignete Segmentierung einer Referenz) enthalte die gleich lange Ausgabesequenz $Y = y_1, \ldots, y_T$ die Markierungen (z.B. die Namen der Metadaten-Felder) für X. Mit einem CRF kann genau die Wahrscheinlichkeit $P(Y|X)$ berechnet werden. Gesucht ist diejenige Ausgabesequenz Y, für die $P(Y|X)$ maximal ist. Peng et al. [48] erreichen unter Verwendung von CRFs eine Genauigkeit von 99,7% korrekt identifizierter Titel und eine Genauigkeit von 95,4% korrekt identifizierter Felder von Referenzen.

2.1.2 Regelbasierte Heuristiken

Alle bisher vorgestellten Ansätze verfolgen das Ziel, mithilfe von Methoden des Maschinellen Lernens aus Volltexten die Metadaten-Felder von Publikationen und von Referenzen zu identifizieren. Das dazu notwendige Beschaffen geeigneter Trainingsdaten und das Anlernen von Modellen kann unter Umständen mühsam und zeitaufwändig sein.

Ansätze mit regelbasierten Heuristiken verzichten deshalb auf Maschinelles Lernen und versuchen, lediglich mit Beobachtungen zu Positionen und zu strukturellen Eigenschaften von Zeichenketten in Publikationen die gewünschten Metadaten-Felder zu identifizieren. Wie Jöran Beel et al. in ihren Ausführungen [28] feststellen, können die Titel von wissenschaftlichen Publikationen mit regelbasierten Heuristiken deutlich schneller, dafür aber mit einer niedrigeren Genauigkeit als mit Methoden des Maschinellen Lernens identifiziert werden. Zur Identifizierung der Titel in PDF-Dateien stützen sich Beel et al. auf eine regelbasierte Heuristik, die diejenige Zeichenkette, die im oberen Drittel der ersten Seite einer Publikation die größte Schriftgröße aufweist, als Titel definiert. Bei Tests an insgesamt 693 PDF-Dateien erreichten Beel et al. damit eine Genauigkeit von 77,9% korrekt identifizierter Titel und eine Laufzeit von im Schnitt 760 Millisekunden (0,76 Sekunden) pro PDF-Datei. Ein Ansatz zur Extraktion sämtlicher Metadaten-Felder mit SVMs dauerte laut Beel et al. durchschnittlich 4,97 Sekunden pro PDF-Datei.

Cortez et al. [50] verwenden hingegen einen wissensbasierten Ansatz, um die Metadaten-Felder von Referenzen zu erkennen. Dazu erstellen sie automatisch eine Wissensbasis aus einer gegebenen Menge von Literaturdatenbank-Einträgen, die für jedes Metadaten-Feld typische Werte identifiziert. Cortez et al. teilen Referenzen an allen Zeichen, die keine Buchstaben oder Zahlen sind, in Blöcke auf und markieren jeden Block mithilfe der erstellten Wissensbasis mit einem Namen eines Metadaten-Feldes. Alle Blöcke, die mit keinem Namen markiert werden konnten, werden unter Berücksichtigung von regelbasierten Heuristiken markiert. Nachdem alle Blöcke mit den gleichen Markierungen vereinigt wurden, sind alle Metadaten-Felder identifiziert. Die Autoren berichten von einer Genauigkeit von über 94% korrekt erkannter Metadaten-Felder mit dieser Methode.

Da uns mit DBLP und Medline zwei Literaturdatenbanken als verifizierende Instanzen für die extrahierten Daten zur Verfügung stehen, unterscheidet sich unser Ansatz zur Extraktion von Metadaten-Feldern von den meisten bisher vorgestellten Ansätzen: In einem ersten Schritt extrahieren wir sowohl den Titel als auch die Referenzen jeweils als Ganzes, ohne die extrahierten Zeichenketten in einzelne Metadaten-Felder zu klassifizieren. Wir können deshalb auf Methoden des Maschinellen Lernens verzichten. Stattdessen greifen wir die Idee von Jöran Beel et al. auf und entwickeln eine regelbasierte Heuristik, die neben dem Titel auch die Referenzen aus wissenschaftlichen Publikationen extrahieren kann. Um trotzdem die Werte der Metadaten-Felder zu erhalten, bestimmen wir für jede extrahierte Zeichenkette in einem wissensbasierten Ansatz ihren repräsentierenden Eintrag in einer Literaturdatenbank. Diesem können wir die gewünschten Werte für die Metadaten-Felder entnehmen.

2.2 Methoden der Metadaten-Zuordnung

Ein naheliegender Ansatz für die Zuordnung einer extrahierten Zeichenkette zu einem Eintrag einer Literaturdatenbank ist die Identifizierung eindeutig charakterisierender Metadaten-Felder innerhalb der Zeichenkette, die zur Suche in einer Literaturdatenbank genutzt werden können. Neben den oben vorgestellten probabilistischen Methoden können dazu ebenfalls regelbasierte Heuristiken verwendet werden. Kratzer [37], der mit einer regelbasierten Heuristik u.a. die Autoren und das Erscheinungsjahr identifiziert und versucht, mit ihnen den repräsentierenden Eintrag in einer Literaturdatenbank zu finden, verwendet solch einen Ansatz. Ein Problem dabei ist jedoch, dass die Korrektheit der extrahierten Daten von entscheidender Bedeutung sind. Enthält die extrahierte Zeichenkette Fehler jeglicher Art (z.B. Extraktionsfehler durch eine fehlerhafte Zeichenkodierung, Schreibfehler von Autoren) oder existieren alternative Schreibweisen für einen Wert eines Metadaten-Feldes, so schlägt die Suche nach einem repräsentierenden Eintrag in einer Literaturdatenbank fehl, obwohl möglicherweise ein entsprechender Eintrag existiert.

Ein weiterer Ansatz ist deshalb die Anwendung einer approximativen Zeichenkettensuche, die im Allgemeinen in der Literatur sehr ausführlich diskutiert wird, da sie in sehr vielen Einsatzgebieten Anwendung findet – vor allem in der Bioinformatik, wo mithilfe von Techniken der approximativen Zeichenkettensuche z.B. Muster in DNA-Strängen gefunden werden. Wir können jeden Eintrag einer Literaturdatenbank als eine Zeichenkette repräsentieren, der sich aus beliebigen Metadaten-Felder zusammensetzt. Wir suchen mithilfe der approximativen Zeichenkettensuche dann diejenige Zeichenkette eines Literaturdatenbank-Eintrags, für die der extrahierte Titel bzw. die extrahierte Referenz am ähnlichsten ist.

Eine Möglichkeit für die Realisierung einer approximativen Zeichenkettensuche ist die Bestimmung der Ähnlichkeit über die *Levenshtein-Distanz* [40] zwischen zwei Zeichenketten X und Y. Die Levenshtein-Distanz bezeichnet die minimal benötigte Anzahl von Operationen, die Zeichenkette X in die Zeichenkette Y zu überführen. Mögliche Operationen sind dabei das Einfügen, das Löschen und das Ersetzen eines Zeichens in X oder Y. Der Algorithmus von Needleman und Wunsch [45] ist eine Methode der dynamischen Programmierung und berechnet die Levenshtein-Distanz in Zeit $O(|X||Y|)$. Mithilfe der dynamischen Programmierung können optimale Lösungen für gegebene Probleme gefunden werden, indem zunächst Lösungen für (kleinere) Teilprobleme berechnet werden: Bezeichne N_i das Präfix der Länge i von einer Zeichenkette N mit $0 \leq i \leq |N|$ und $N[j]$ das Zeichen an Position i in N mit $1 \leq j \leq |N|$. Die Levenshtein-Distanz $d_L(X, Y) = d_L(X_{|X|}, Y_{|Y|})$ zwischen den Zeichenketten X und Y können wir wie folgt berechnen:

$$
\begin{aligned}
d_L(X_0, Y_0) &= 0 \\
d_L(X_i, Y_0) &= i \\
d_L(X_0, Y_j) &= j \\
d_L(X_i, Y_j) &= \min \begin{cases}
d_L(X_i, Y_{j-1}) + 1 & \text{Einfügen des Zeichens } Y[j] \\
d_L(X_{i-1}, Y_j) + 1 & \text{Löschen des Zeichens } X[i] \\
d_L(X_{i-1}, Y_{j-1}) + 1 & \text{Ersetzen des Zeichens } X[i] \text{ durch } Y[j] \\
d_L(X_{i-1}, Y_{j-1}) + 0 & \text{wenn } X[i] = Y[j]
\end{cases}
\end{aligned}
$$

In Kapitel 4.3 werden wir diesen Algorithmus noch einmal aufgreifen und näher erläutern.

Ein Nachteil der dynamischen Programmierung ist die hohe Anzahl (teilweise unnötiger) Berechnungen: Für alle Paare möglicher Präfixe von X und Y berechnen wir die jeweilige Levenshtein-Distanz, obwohl dies nur für einen Bruchteil von Paaren notwendig wäre. Für den Fall, dass zwischen X und Y höchstens k Unterschiede ($0 \leq k \leq \max(|X|, |Y|)$) auftreten dürfen, konnten Landau und Vishkin [38] die Anzahl solcher Berechnungen reduzieren und die Laufzeit auf $O(k * \max(|X|, |Y|))$ verringern. Sie kombinieren die Methode der dynamischen Programmierung hierzu mit *Suffixbäumen*. Mit einem Suffixbaum über eine Zeichenkette N, der alle Suffixe von N speichert, ist unter anderem eine effiziente Suche in N möglich.

Die Idee von Landau und Vishkin ist, die dynamische Programmierung durch die Berechnung des längsten, gemeinsamen Präfixes von den Zeichenketten X und Y, welche mit Suf-

fixbäumen in Zeit $O(1)$ berechnet werden können, zu beschleunigen. Dadurch können viele Berechnungen zwischen Teilzeichenketten aus X und Y eingespart werden. Zusätzlich kann die Berechnung der Levenshtein-Distanz abgebrochen werden, sobald sie den Wert von k überschreitet. Die Berechnung der Levenshtein-Distanz zwischen den Zeichenketten X und Y kann somit allgemein beschleunigt werden.

2.3 Verwandte Software

Zur Untersuchung von verwandter Software können wir sowohl das Konzept von Literatursuchmaschinen als auch das Konzept von Literaturverwaltungsprogrammen heranziehen. Beide Konzepte haben Aufgaben, die mit unseren Zielen teilweise übereinstimmen.

2.3.1 Literatursuchmaschinen

Literatursuchmaschinen sind spezielle Suchmaschinen, mit denen z.B. Literaturdatenbanken nach wissenschaftlichen Publikationen durchsucht werden können. Neben den Metadaten und den Volltexten können mit ausgewählten Literatursuchmaschinen auch die Referenzen wissenschaftlicher Publikationen durchsucht werden. Derartige Suchmaschinen, wie z.B. *CiteSeer$^{\chi}$* [4], *Google Scholar* [10], *ACM Digital Library* [1] oder *IEEE Xplore* [13] müssen deshalb in der Lage sein, die Metadaten und Referenzen aus Publikationen zu extrahieren. Die prinzipielle Vorgehensweise sei im Folgenden am Beispiel von CiteSeer$^{\chi}$ erläutert:

CiteSeer$^{\chi}$

Mit der ursprünglich *CiteSeer* genannten Literatursuchmaschine können die Volltexte und Referenzen frei verfügbarer wissenschaftlicher Publikationen durchsucht werden. Die Referenzen werden mit einem sogenannten *Autonomous Citation Indexing* (ACI) automatisch indiziert. Dazu sucht das ACI-System im Internet (z.B. in Mailinglisten oder in Newsgroups) automatisch nach neuen Publikationen [29]. Aus jeder gefundenen PDF- und Postscript-Datei wird der Volltext extrahiert und der Titel, die Autoren sowie alle Metadaten-Felder der Referenzen identifiziert.

Ursprünglich wurden diese Daten u.a. anhand von Schriftgrößen, Referenzbezeichnern (z.B. „[6]“, „[Giles97]“, etc.), vertikalen Zeilenabständen, Interpunktionen oder Einrückungen identifiziert [32, 39]. Nach einer umfassenden Neustrukturierung der Suchmaschine, die nötig wurde, um den stetig wachsenden Benutzerzahlen gerecht zu werden [43], werden diese Daten nun mithilfe von Support Vector Machines [34] identifiziert. Im Zuge dieser Neustrukturierung wurde CiteSeer in *CiteSeer$^{\chi}$* umbenannt.

Für eine gegebene Publikation P kann mit CiteSeer$^{\chi}$ dank der Indizierung der Referenzen sowohl eine sogenannte *Rückwärtssuche* (welche Publikationen wurden von P referenziert?) als auch eine *Vorwärtssuche* (von welchen Publikationen wurde P referenziert?) durchgeführt werden. CiteSeer$^{\chi}$ stellt damit geeignete Werkzeuge zur Verfügung, auf einfache Weise ein möglichst breites Spektrum von Publikationen zu einem bestimmen Themengebiet zu finden.

2.3.2 Literaturverwaltungsprogramme

Die Hauptaufgabe von Literaturverwaltungsprogrammen ist die Sammlung und Verwaltung wissenschaftlicher Publikationen, um Benutzer bei einer Literaturrecherche und bei der Erstellung von Literaturverzeichnissen für eigene Veröffentlichungen zu unterstützen. Die Existenz verschiedener Literaturverwaltungsprogramme liegt vor allem in ihren unterschiedlichen Funktionsumfängen begründet:

- **Mendeley:** Mit Mendeley [17] können die Metadaten (Titel, Autoren, Erscheinungsjahr, etc.) aus PDF-Dateien wissenschaftlicher Publikationen automatisch extrahiert werden, die zusammen mit der Publikation in einer Bibliothek aufgelistet werden. Alle Publikationen können mit ihren Volltexten durchsucht werden und in einem internen PDF-Betrachter gelesen werden. Aus den Publikationen können zudem Literaturverzeichnisse

für eigene Veröffentlichungen automatisch erstellt werden. Bei Mendeley handelt es sich um eine Kombination aus Desktop- und Webanwendung und ist für die Betriebssysteme Windows, Linux und Mac verfügbar. Über eine Webschnittstelle werden dem Benutzer weitere Publikationen vorgeschlagen und können die Benutzer Publikationen untereinander austauschen. Auch deshalb sieht sich Mendeley als „Last.fm der Wissenschaft“[1] [35].

- **Zotero:** Zotero [25] ist eine Erweiterung für den Webbrowser *Firefox*. Auch mit Zotero können die Titel und Autoren aus Publikationen extrahiert werden, die mit Metadaten z.B. aus *Google Scholar* [10] abgeglichen und erweitert werden. Ähnlich zu Mendeley können Literaturverzeichnisse erstellt und Volltexte der Publikationen durchsucht werden. Alle Daten und Dateien werden lokal auf dem Rechner des Benutzers gespeichert und können auf Wunsch mit einem persönlichen Onlinekonto synchronisiert werden, um von jedem Rechner mit Internetanschluss aus auf die Daten zugreifen zu können.

- **Citavi:** Ein weiteres Literaturverwaltungsprogramm ist Citavi [3], das als reine (lizenzpflichtige) Desktopanwendung konzipiert wurde und bisher nur für Windows verfügbar ist (Stand: September 2011). Neben den bereits bekannten Funktionalitäten können mit Citavi u.a. Online-Recherchen in Literaturkatalogen und -datenbanken durchgeführt werden. Gewünschte Daten können in eine eigene Bibliothek importiert werden.

Ähnliche Literaturverwaltungsprogramme, wie z.B. *CiteULike* [5], *JabRef* [15] oder *EndNote* [9] bieten größtenteils vergleichbare Funktionsumfänge und sollen hier deshalb nur aus Gründen der Vollständigkeit erwähnt werden.

Keines der zahlreichen Programme konnte jedoch die Referenzen aus Publikationen extrahieren. Ursprünglich enthielt zwar Mendeley diese Funktionalität; diese wurde jedoch nach eigenen Angaben wieder deaktiviert, da sie wohl zuviel (Speicher-)Ressourcen verbraucht und keine zuverlässigen Ergebnisse geliefert hat [16].

Wir sehen also Bedarf an einem Programm, das alle notwendigen Schritte einer individuellen Literaturrecherche aggregiert und so einfach und komfortabel wie möglich gestaltet. Zu diesen Schritten gehören die Sammlung und Verwaltung wissenschaftlicher Publikationen, das schnelle Identifizieren von Publikationen anhand automatisch bestimmter Metadaten, das Durchsuchen der Metadaten und Volltexte von Publikationen, das Durchsuchen von (externen) Literaturdatenbanken, das Betrachten der PDF-Dateien der Publikationen sowie das Finden verwandter Publikationen. Obwohl es eine wesentliche Arbeitserleichterung und Zeitersparnis bedeuten würde, existiert ein solches Programm in diesem Umfang bisher nicht. Wie bereits vorgestellt, existieren zwar zahlreiche Literaturverwaltungsprogramme, die aber jeweils nur einen Teil der genannten Anforderungen abdecken. Insbesondere fehlt die Möglichkeit, aus Literaturverzeichnissen gegebener Publikationen die Referenzen extrahieren zu können und damit das Potential, verwandte Publikationen auf einfache Weise zu finden.

Literatursuchmaschinen wie z.B. CiteSeer$^{\chi}$ [4] bieten zwar die Möglichkeit, die Referenzen von frei verfügbaren Publikationen zu durchsuchen, dafür bieten sie aber nicht die Vorteile eines Literaturverwaltungsprogramms. Da wir aber darin eine großartige Möglichkeit sehen, den Ablauf einer Literaturrecherche erheblich zu vereinfachen und die dafür benötigte Zeit auf ein Minimum zu reduzieren, haben wir uns dazu entschlossen, solch ein System zu implementieren.

[1] Last.fm ist eine Webseite, die einem Benutzer, basierend auf seinen angehörten Radiosendern oder Musikstücken, neue Musik vorschlägt: `http://www.last.fm/`.

Kapitel 3

Extraktion von Titel & Referenzen

In diesem Kapitel erklären wir unsere Vorgehensweise zur Extraktion des Titels und der Referenzen aus PDF-Dateien wissenschaftlicher Publikationen. Die Schwierigkeit dabei ist, dass es keine allgemeingültigen Normen gibt, den Titel und die Referenzen zu positionieren und zu strukturieren. Um eine möglichst große Anzahl korrekter Ergebnisse zu erhalten, müssen wir also viele verschiedende Titel- und Referenzstile berücksichtigen. Wir möchten dabei den Titel und jede Referenz als Ganzes extrahieren, jedoch nicht die jeweiligen Metadaten-Felder identifizieren. Es ist uns deshalb möglich, für beide Fälle eine regelbasierte Heuristik zu verwenden und auf probabilistische Methoden, wie wir sie im vorigen Kapitel vorgestellt haben, zu verzichten. Das bedeutet, dass wir den Titel und die Referenzen lediglich anhand ihrer strukturellen Informationen identifizieren, ohne notwendige Hilfsstrukturen aufwändig erstellen und anlernen zu müssen.

Wir werden zunächst das allgemeine Prinzip der Text-Extraktion aus PDF-Dateien vorstellen und im Anschluss erläutern, wie wir innerhalb des extrahierten Textes den Titel und die Referenzen identifizieren können.

3.1 Allgemeines Prinzip der Text-Extraktion

Zur Extraktion des Textinhaltes aus PDF-Dateien verwenden wir die frei verfügbare Java-Bibliothek *Apache PDFBox* [2]. PDFBox extrahiert den Text einer PDF-Datei Zeichen für Zeichen und speichert jedes in einem Objekt `TextPosition` ab, welches man sich als ein Rechteck, das genau ein Zeichen umschließt, vorstellen kann (*Bounding Box*). Jede j-te `TextPosition` $TP_{i,j}$ aus der i-ten Textzeile l_i ($1 \leq i \leq n$ und $1 \leq j \leq m_i$; $n =$ Anzahl der Textzeilen in der PDF-Datei und $m_i =$ Anzahl `TextPosition`-Objekte in Zeile i) wird über die X-Koordinate $x(TP_{i,j})$ und die Y-Koordinate $y(TP_{i,j})$ der linken, oberen Ecke der Bounding Box, der Höhe $h(TP_{i,j})$ und Breite $w(TP_{i,j})$ der Bounding Box sowie über die Schriftart $fn(TP_{i,j})$ und die Schriftgröße $fs(TP_{i,j})$ des enthaltenen Zeichens definiert (vgl. Schritt 1 in Beispiel 3.1).

Die von PDFBox erhaltenen Daten sind nicht immer vollkommen exakt, d.h. Werte, die logisch äquivalent sein müssten, können tatsächlich um einen minimalen Toleranzwert abweichen. In Beispiel 3.1 stimmen z.B. die Y-Koordinaten der erhaltenen `TextPosition`-Objekte nicht alle überein, obwohl sich die Objekte alle in der gleichen Textzeile befinden. Wenn wir im weiteren Verlauf unserer Ausführungen Werte von `TextPosition`-Objekten miteinander vergleichen, implizieren wir deshalb immer, dass wir eine Toleranz berücksichtigen, die sich üblicherweise zwischen 0 und 0,5 bewegt. Liegt die Differenz $a - b$ zwischen zwei Werten a und b innerhalb des Intervalls $[-0.5, 0.5]$, so erachten wir a und b als äquivalent. Wenn hingegen $(a - b) > 0,5$, so gilt $a > b$ und wenn $(a - b) < -0,5$, so gilt $a < b$.

Wir setzen die erhaltenen `TextPosition`-Objekte anhand ihrer Attribute wieder zu Wörtern zusammen, indem wir dazu die (horizontalen) Abstände der `TextPosition`-Objekte zueinander betrachten: Liegt der Abstand zwischen $x(TP_{i,j})$ (wobei $j > 1$) und der X-Koordinate $x_r(TP_{i,j-1}) = x(TP_{i,j-1}) + w(TP_{i,j-1})$ des rechten Randes von der vorherigen `TextPosition`

Accurate Information Extraction from Research Papers

⇓ (1)

Accurate Information Extraction from Research Papers

```
TextPosition['A': x:135,00, y:90,84, w:10,3, h:9,89, fs:14,34, fn:Times-Bold]
TextPosition['c': x:145,31, y:90,85, w:6,36, h:6,98, fs:14,34, fn:Times-Bold]
TextPosition['c': x:151,67, y:90,84, w:6,36, h:6,98, fs:14,34, fn:Times-Bold]
TextPosition['u': x:158,03, y:90,84, w:7,97, h:6,81, fs:14,34, fn:Times-Bold]
TextPosition['r': x:165,95, y:90,86, w:6,36, h:6,78, fs:14,34, fn:Times-Bold]
TextPosition['a': x:172,31, y:90,84, w:7,17, h:6,98, fs:14,34, fn:Times-Bold]
TextPosition['t': x:179,51, y:90,84, w:4,77, h:9,21, fs:14,34, fn:Times-Bold]
TextPosition['e': x:184,31, y:90,83, w:6,36, h:6,98, fs:14,34, fn:Times-Bold]
TextPosition['I': x:194,39, y:90,84, w:5,58, h:9,69, fs:14,34, fn:Times-Bold]
TextPosition['n': x:200,03, y:90,79, w:7,97, h:6,78, fs:14,34, fn:Times-Bold]
TextPosition['f': x:207,95, y:90,83, w:4,77, h:9,91, fs:14,34, fn:Times-Bold]
TextPosition['o': x:212,39, y:90,84, w:7,17, h:6,98, fs:14,34, fn:Times-Bold]
                                    ...
TextPosition['P': x:434,87, y:90,84, w:8,76, h:9,69, fs:14,34, fn:Times-Bold]
TextPosition['a': x:443,51, y:90,82, w:7,17, h:6,98, fs:14,34, fn:Times-Bold]
TextPosition['p': x:450,71, y:90,84, w:7,97, h:9,72, fs:14,34, fn:Times-Bold]
TextPosition['e': x:458,63, y:90,81, w:6,36, h:6,98, fs:14,34, fn:Times-Bold]
TextPosition['r': x:464,99, y:90,84, w:6,36, h:6,78, fs:14,34, fn:Times-Bold]
TextPosition['s': x:471,35, y:90,85, w:5,58, h:6,98, fs:14,34, fn:Times-Bold]
```

⇓ (2)

Accurate Information Extraction from Research Papers

```
TextLine["...": x:135,00, y:90,84, w:341,93, h:9,91, fs:14,34, fn:times-bold]
```

Beispiel 3.1: Die Zusammensetzung der Textzeile *„Accurate Information Extraction"* aus den von PDFBox erhaltenen `TextPosition`-Objekten. Jedes `TextPosition`-Objekt enthält Angaben zu seiner X-Koordinate x, Y-Koordinate y, Breite w, Höhe h, Schriftgröße fs und Schriftart fn (Schritt 1). Aus diesen Angaben berechnen wir entsprechende Werte für jede Textzeile (Schritt 2).

$TP_{i,j-1}$ innerhalb einer definierten Toleranz, so fügen wir das Zeichen dem aktuellen Wort hinzu. Anderenfalls fügen wir ein Leerzeichen ein und beginnen mit $TP_{i,j}$ ein neues Wort.

Weil wir den Text von PDF-Dateien künftig zeilenweise analysieren wollen, berechnen wir mithilfe der oben genannten Merkmale aller `TextPosition`-Objekte $TP_{i,j}$ äquivalente Werte für die Textzeile l_i (vgl. Schritt 2 in Beispiel 3.1). Aus ihnen ergeben sich die X- und Y-Koordinaten $x(l_i)$ und $y(l_i)$, die Höhe $h(l_i)$, die Breite $w(l_i)$, die Schriftart $fn(l_i)$ und die Schriftgröße $fs(l_i)$ der Textzeile l_i wie folgt:

- **X- und Y-Koordinate:** Die X-Koordinate $x(l_i)$ und die Y-Koordinate $y(l_i)$ beschreiben die linke, obere Ecke der Bounding Box von der i-ten Textzeile. Da das erste `TextPosition`-Objekt $TP_{i,1}$ zugleich auch das linkeste von l_i ist, können wir für die X-Koordinate $x(l_i)$ und die Y-Koordinate $y(l_i)$ die entsprechenden Werte von $TP_{i,1}$ übernehmen. Es gilt also:

$$x(l_i) = x(TP_{i,1})$$
$$y(l_i) = y(TP_{i,1})$$

- **Höhe** Die Höhe $h(l_i)$ der Textzeile l_i wird durch das höchste `TextPosition`-Objekt innerhalb von l_i vorgegeben. Es gilt:

$$h(l_i) = \max_{1 \leq k \leq m_i} (h(TP_{i,k}))$$

- **Breite** Zur Berechnung der Breite $w(l_i)$ einer Textzeile l_i genügt es nicht, die Breiten aller `TextPosition`-Objekte $TP_{i,j} \in l_i$ aufzusummieren, denn dann hätten wir die Breiten der

eingefügten Leerzeichen vernachlässigt. Stattdessen messen wir die Breite über die Distanz zwischen dem linken Rand der ersten `TextPosition` $TP_{i,1}$ und dem rechten Rand der letzten `TextPosition` TP_{i,m_i} einer Textzeile l_i:

$$w(l_i) = x(TP_{i,m_i}) + w(TP_{i,m_i}) - x(TP_{i,1})$$

- **Schriftgröße** Im Idealfall ist die Schriftgröße aller `TextPosition`-Objekte einer Textzeile l_i gleich und wir können diese für die Schriftgröße $fs(l_i)$ übernehmen. Weitaus weniger offensichtlich ist die Berechnung von $fs(l_i)$, wenn sich die Schriftgrößen der Zeichen innerhalb der Textzeile voneinander unterscheiden. Hierbei genügt ein Zeichen, dass sich in seiner Schriftgröße von den anderen Zeichen unterscheidet, um dieses Szenario herbeizuführen. Wenn wir nun für $fs(l_i)$ die minimale oder maximale Schriftgröße innerhalb von l_i übernehmen würden, hätte solch ein Zeichen Einfluss auf die gesamte Textzeile. Als Alternative berechnen wir $fs(l_i)$ deshalb aus dem arithmetischen Mittel der Schriftgrößen aller `TextPosition`-Objekte aus der Textzeile l_i:

$$fs(l_i) = \frac{\sum_{k=1}^{m_i} fs(TP_{i,k})}{m_i}$$

- **Schriftart** Ähnlich wie bei der Berechnung der Schriftgröße, können sich Zeichen einer Textzeile auch in ihrer Schriftart voneinander unterscheiden. Im Gegensatz dazu können wir aber für die Bestimmung der Schriftart $fn(l_i)$ einer Textzeile l_i kein arithmetisches Mittel über die Schriftarten bilden. Wir wählen deshalb die „dominanteste“ Schriftart in l_i für $fn(l_i)$ aus, d.h. diejenige Schriftart, die am häufigsten in l_i vorkommt. Formal formuliert:

  Sei $\mathbb{F}_i$ die Menge aller in l_i vorkommenden Schriftarten und $\mathbb{T}_{i,f_k}$ die Menge aller Zeichen in l_i mit Schriftart f_k (mit $f_k \in \mathbb{F}_i$). Dann gilt:

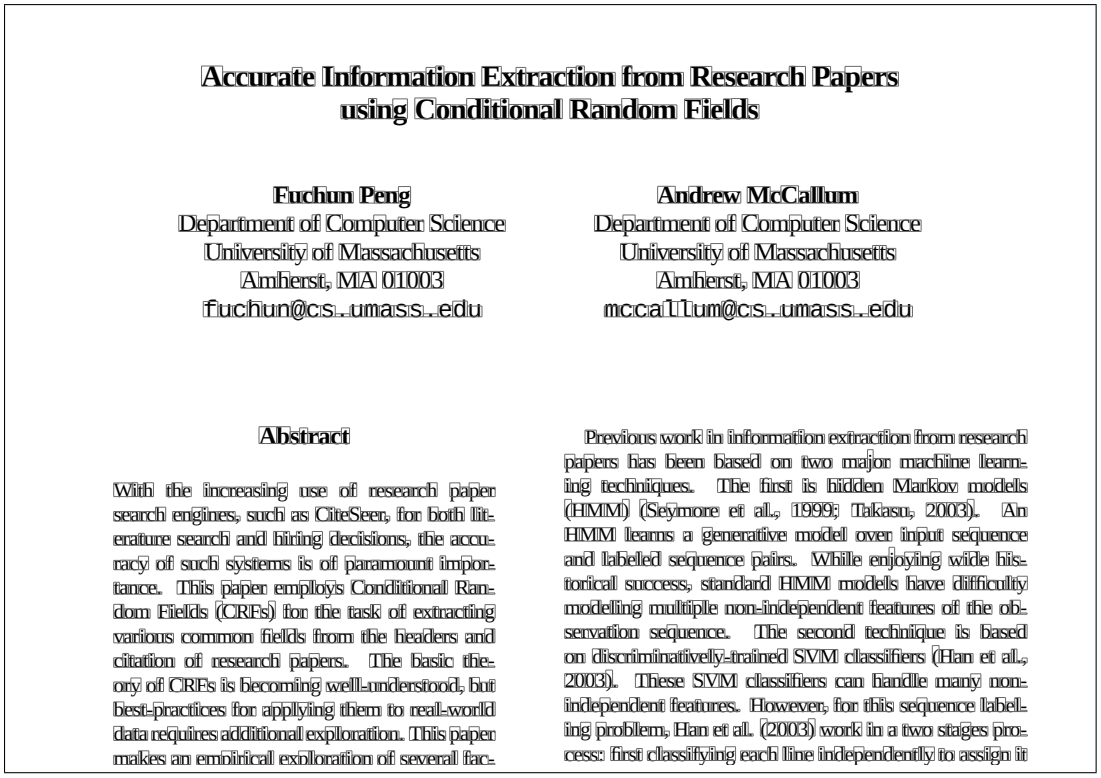
$$fn(l_i) = \arg\max_{f_k \in \mathbb{F}_i} \left(\frac{|\mathbb{T}_{i,f_k}|}{m_i} \right)$$

Abbildung 3.2 fasst das Prinzip der Text-Extraktion und der Zusammensetzung der Textzeilen noch einmal anhand einer ersten Seite einer wissenschaftlichen Publikation zusammen. Alle beschriebenen Berechnungen führen wir für genau so viele Textzeilen aus, wie sie für die Titel-Extraktion bzw. die Referenzen-Extraktion nötig sind. Die berechneten Attribute dienen vor allem dazu, wichtige Strukturen zu erkennen und um den Titel und die Referenzen einer wissenschaftlichen Publikation zu identifizieren. Wie wir in den nächsten Kapiteln sehen werden, ist dabei die Korrektheit der Daten für die erfolgreiche Ausführung beider Aufgaben von entscheidender Bedeutung.

3.2 Extraktion von Titeln

Der Titel einer wissenschaftlichen Publikation fasst das Thema der Publikation kurz und prägnant zusammen und ist damit das wichtigste Merkmal, das im Kontext dieser Arbeit zur eindeutigen Identifizierung einer Publikation verwendet wird. Für die Titel-Extraktion machen wir uns dabei die Tatsache zu Nutze, dass der Titel sich deshalb meist am Anfang einer Publikation befindet und durch eine besondere *Schriftauszeichnung* hervorgehoben ist.

Definition (Schriftauszeichnung & Schriftschnitt)**.** *Der Begriff der Schriftauszeichnung sei ein Oberbegriff für die typografischen Eigenschaften eines Textes, wie die Schriftart, die Schriftgröße und der Schriftschnitt. Der Schriftschnitt eines Textes sei ein Oberbegriff für die Schriftstärke (normal, fett, etc.) und die Schriftlage (normal, kursiv, etc.) [21].*



Accurate Information Extraction from Research Papers using Conditional Random Fields

Fuchun Peng
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
fuchun@cs.umass.edu

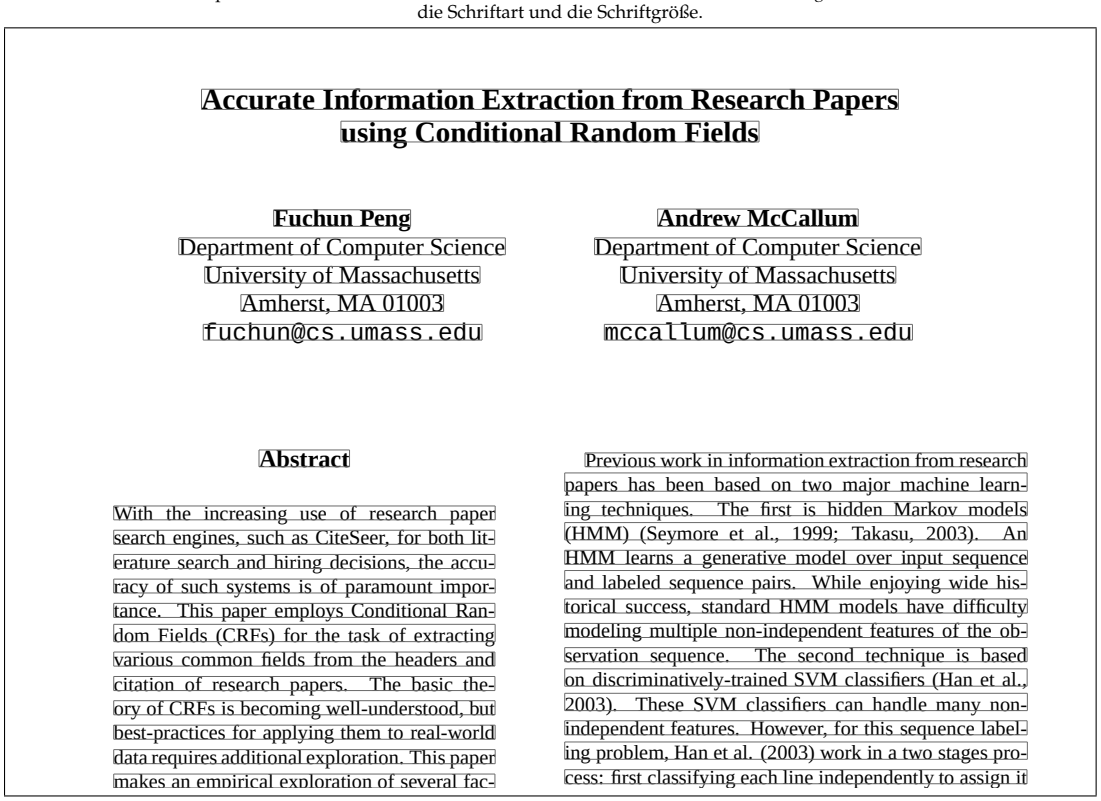
Andrew McCallum
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
mccallum@cs.umass.edu

Abstract

With the increasing use of research paper search engines, such as CiteSeer, for both literature search and hiring decisions, the accuracy of such systems is of paramount importance. This paper employs Conditional Random Fields (CRFs) for the task of extracting various common fields from the headers and citation of research papers. The basic theory of CRFs is becoming well-understood, but best-practices for applying them to real-world data requires additional exploration. This paper makes an emprirical exploration of several fac-

Previous work in information extraction from research papers has been based on two major machine learning techniques. The first is hidden Markov models (HMM) (Seymore et al., 1999; Takasu, 2003). An HMM learns a generative model over input sequence and labeled sequence pairs. While enjoying wide historical success, standard HMM models have difficulty modeling multiple non-independent features of the observation sequence. The second technique is based on discriminatively-trained SVM classifiers (Han et al., 2003). These SVM classifiers can handle many non-independent features. However, for this sequence labelling problem, Han et al. (2003) work in a two stages process: first classifying each line independently to assign it

(a) PDFBox liefert uns den Textinhalt einer PDF-Datei Zeichen für Zeichen als `TextPosition`-Objekte. Jede `TextPosition` speichert die X- und Y-Koordinaten der linken oberen Ecke seiner *Bounding Box*, die Höhe, die Breite, die Schriftart und die Schriftgröße.



Accurate Information Extraction from Research Papers using Conditional Random Fields

Fuchun Peng
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
fuchun@cs.umass.edu

Andrew McCallum
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
mccallum@cs.umass.edu

Abstract

With the increasing use of research paper search engines, such as CiteSeer, for both literature search and hiring decisions, the accuracy of such systems is of paramount importance. This paper employs Conditional Random Fields (CRFs) for the task of extracting various common fields from the headers and citation of research papers. The basic theory of CRFs is becoming well-understood, but best-practices for applying them to real-world data requires additional exploration. This paper makes an empirical exploration of several fac-

Previous work in information extraction from research papers has been based on two major machine learning techniques. The first is hidden Markov models (HMM) (Seymore et al., 1999; Takasu, 2003). An HMM learns a generative model over input sequence and labeled sequence pairs. While enjoying wide historical success, standard HMM models have difficulty modeling multiple non-independent features of the observation sequence. The second technique is based on discriminatively-trained SVM classifiers (Han et al., 2003). These SVM classifiers can handle many non-independent features. However, for this sequence labeling problem, Han et al. (2003) work in a two stages process: first classifying each line independently to assign it

(b) Wir setzen diese `TextPosition`-Objekte zu Textzeilen zusammen und berechnen entsprechende Werte für jede Textzeile.

Abbildung 3.2: Das allgemeine Prinzip der Textzeilen-Extraktion aus PDF-Dateien im Überblick.

(a) (b) (c)

Abbildung 3.3: Die Titelseiten von drei verschiedenen wissenschaftlichen Publikationen.

Da in der Regel aber keine allgemeingültigen Aussagen über die Position und die Schriftauszeichnung des Titels gemacht werden können, erstellen wir eine Heuristik, mit der wir, basierend auf Beobachtungen zu Positionen und Schriftauszeichnungen von Titeln in wissenschaftlichen Publikationen, möglichst viele Titel korrekt identifizieren können.

In Abbildung 3.3 sind die Titelseiten von drei wissenschaftlichen Publikationen abgebildet. In allen drei Publikationen unterscheiden sich die Titel entweder in ihrer Schriftauszeichnung oder ihrer Position: So muss der Titel keineswegs nur aus einer Zeile bestehen (vgl. Abbildungen 3.3a und 3.3c) und nicht unbedingt in der ersten Textzeile beginnen (vgl. Abbildungen 3.3b und 3.3c). Auch ist die Ausrichtung des Titels nicht fest vorgegeben: Während der Titel in den Abbildungen 3.3a und 3.3b zentriert ist, ist der Titel in Abbildung 3.3c linksbündig ausgerichtet. Dennoch können wir auch einige Gemeinsamkeiten der Titel beobachten, wie z.B. folgende typische Eigenschaften:

- **Position:** Bei allen drei Publikationen befindet sich der Titel auf der ersten Seite.
- **Schriftauszeichnung:** Der Titel ist in allen drei Publikationen durch eine andere Schriftart, einem anderen Schriftschnitt oder eine größere Schriftgröße gegenüber dem restlichen Text hervorgehoben. Die Schriftauszeichnung bleibt innerhalb des Titels konsistent.
- **Titellänge:** Alle drei Titel erreichen eine gewisse Mindestlänge, d.h. wir können einen Schwellenwert definieren, so dass alle Titel jeweils länger als dieser Schwellenwert sind.

Aus diesen Eigenschaften werden wir nun eine regelbasierte Heuristik erstellen, mit der wir den Titel innerhalb der extrahierten Textzeilen identifizieren können. Auch wenn die Eigenschaften für die meisten Titel wissenschaftlicher Publikationen zutreffend sein werden, sind sie dennoch keineswegs allgemeingültig. Wir müssen also damit rechnen, dass es Publikationen gibt, für dessen Titel diese Eigenschaften nicht gelten und wir sie deshalb nicht identifizieren können. Unsere Experimente, in denen wir u.a. die Titel-Extraktion evaluieren (siehe Kapitel 5), werden letztendlich Aufschluss darüber geben, wie viele Titel wir mit unserer Heuristik identifizieren können.

Eine Heuristik zur Identifizierung von Titeln

Wir setzen die beobachteten Eigenschaften eines Titels in eine Heuristik um, indem wir die Extraktion der Zeichen auf die erste Seite der Publikation beschränken. Wir setzen die Zeichen wie in Kapitel 3.1 besprochen zu Textzeilen zusammen und unterteilen sie in verschiedene Regionen, so dass jede Region alle unmittelbar aufeinander folgenden Textzeilen mit der gleichen Schriftauszeichnung enthält. Damit erhalten wir eine Gruppierung, wie sie beispielhaft in Abbildung 3.4 dargestellt ist.

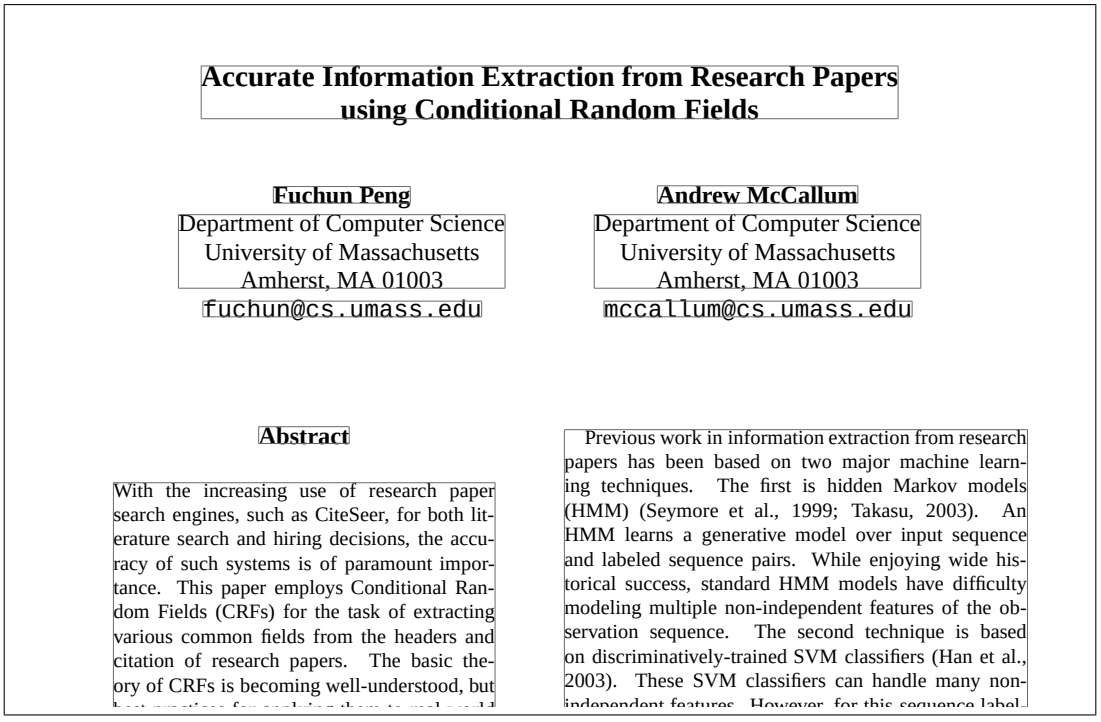


Abbildung 3.4: Einteilung der Textzeilen in Regionen. Jede Region enthält alle aufeinander folgenden Textzeilen mit der gleichen Schriftauszeichnung.

Wir gehen in unserer Heuristik davon aus, dass die Schriftauszeichnung innerhalb des Titels konsistent bleibt und sich gleichzeitig von den Schriftauszeichnungen des restlichen Textes unterscheidet. Mit den gewählten Einteilungskriterien gibt es also eine Region, die genau den gesuchten Titel enthält. Da ein Titel am Anfang der Publikation befindet, vermuten wir, dass sich diese Region unter den ersten zehn Regionen befindet. Um sie zu ermitteln, sortieren wir die ersten 10 Regionen, die einen Text mit mindestens der Länge des definierten Schwellenwertes (wir verwenden einen Schwellenwert von 20 Zeichen) enthalten, nach ihrer Schriftauszeichnung. Konkret sortieren wir die Textzeilen absteigend nach ihrer Schriftgröße. Alle Regionen mit der gleichen Schriftgröße sortieren wir anschließend absteigend nach ihrem Schriftschnitt – hierbei gilt folgende Ordnung:

fett gedruckter Text > *kursiv* gedruckter Text > normal gedruckter Text

Sollten Regionen danach immer noch nicht eindeutig sortierbar sein, ordnen wir diese nach ihrer üblichen Lesereihenfolge (also entsprechend ihrer X- und Y-Koordinaten).

Wir bestimmen die in der nun ersten Region enthaltenen Zeichenkette als Titel der vorliegenden wissenschaftlichen Publikation. Anhand diesem werden wir später den repräsentierenden Eintrag der Publikation in einer Literaturdatenbank suchen. Auf die gleiche Weise gehen wir bei den Referenzen vor, deren Extraktion wir im folgenden Abschnitt erklären.

3.3 Extraktion von Referenzen

Im Gegensatz zur Titel-Extraktion ist die Extraktion der Referenzen einer wissenschaftlichen Publikation wesentlich schwieriger, was in der Komplexität der Aufgabe begründet liegt.

Einerseits können das Literaturverzeichnis und die darin enthaltenen Referenzen beliebig positioniert und strukturiert sein. Aufgrund des Fehlens allgemeingültiger Normen für die Strukturierung von Referenzen müssen wir für die Erkennung von Referenzen viele verschiedende Referenz-Stile berücksichtigen, die im Vergleich zu einem Titel weniger offensichtliche Identifizierungsmerkmale aufweisen können. Wegen möglichen nachfolgenden Anhängen wie

Abbildungs- oder Tabellenverzeichnissen kann ein Literaturverzeichnis innerhalb einer Publikation außerdem deutlich flexibler positioniert sein und muss nicht unbedingt der letzte Teil einer Publikation sein.

Andererseits müssen sehr viel mehr Inhalte extrahiert werden, weshalb generell mit mehr Störfaktoren (z.B. mit ungenauen oder fehlerhaften Angaben der `TextPosition`-Objekte) zu rechnen ist, die das Ergebnis der Referenzen-Extraktion beeinflussen können. Trotz dieser Voraussetzungen stellen wir uns die Aufgabe, das Literaturverzeichnis und die darin enthaltenen Referenzen für möglichst viele Publikationen korrekt zu identifizieren.

In den nun folgenden Ausführungen werden wir zu einer Textzeile l_i zusätzlich die Textzeilen l_{i-1} und l_{i+1} zum Vergleich betrachten (sofern diese existieren). Wir verarbeiten die Textzeile l_i, sobald die Textzeile l_{i+1} zusammengesetzt wurde. Im Gegensatz zur Titel-Extraktion, bei der wir zunächst alle Textzeilen der ersten Seite extrahieren, um in ihnen den Titel identifizieren zu können, erfolgt die Referenzen-Extraktion also „*on the fly*“. Die Textzeile l_{i+1} stellt unser einziges Wissen über den der Textzeile l_i folgenden Inhalt dar.

3.3.1 Vorbereitungen für die Identifizierung von Referenzen

Da wir eine Suche nach dem Literaturverzeichnis nicht auf bestimmte Seiten einer Publikation eingrenzen können, durchlaufen wir alle Textzeilen der PDF-Datei sequentiell von Beginn an und überprüfen für jede Textzeile, ob sie die Überschrift des Literaturverzeichnisses enthält. Die Textzeile l_i ist genau dann die Überschrift des Literaturverzeichnisses, wenn sie z.B. eines der folgenden Schlüsselwörter enthält:

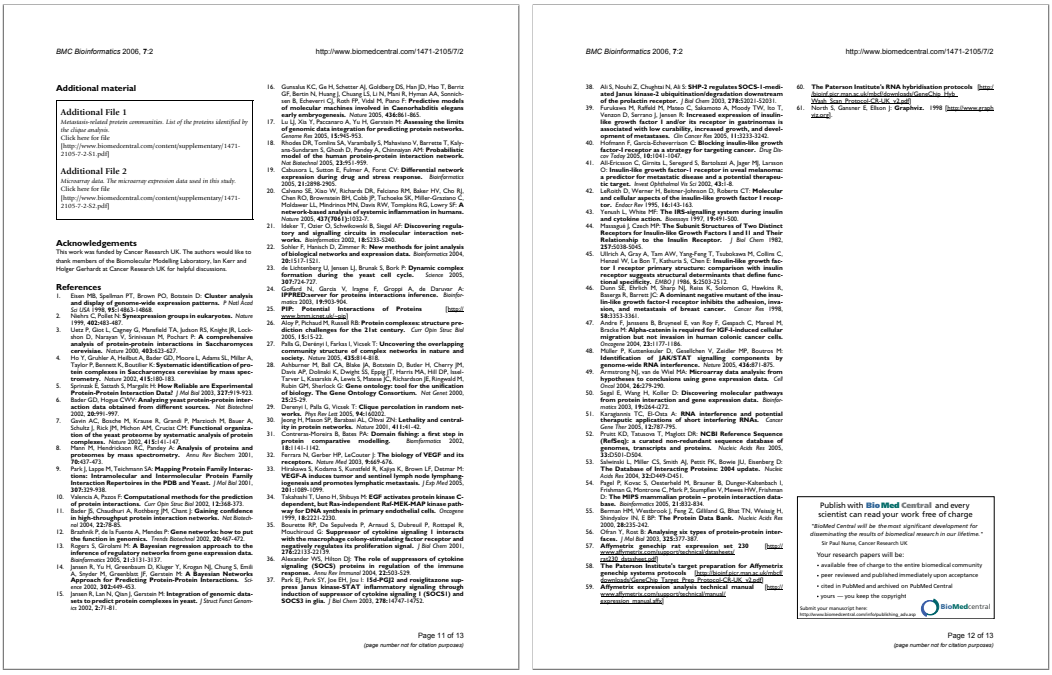
```
References, Literature, Bibliography, References and Notes, Reference,
    Literaturverzeichnis, Literatur, Bibliografie, Bibliographie
```

Mithilfe regulärer Ausdrücke identifizieren wir auch diejenigen Überschriften, denen entsprechende Kapitelnummerierungen vorangestellt sind. Dieser Ansatz impliziert, dass wir das Literaturverzeichnis nicht erkennen, wenn entweder keine Überschrift für das Literaturverzeichnis existiert oder wenn für sie keines der genannten Schlagwörter verwendet wird. Eine alternative Herangehensweise wäre die direkte Erkennung der Referenzen, z.B. anhand von verwendeten Strukturmustern der Referenzen, die aber im Rahmen dieser Masterarbeit nicht weiter verfolgt werden soll.

Berechnung des durchschnittlichen Zeilenabstandes Solange wir die Überschrift des Literaturverzeichnisses nicht erreicht haben, bestimmen wir für jede betrachtete Textzeile l_i mit $i > 1$ ihren Zeilenabstand $d_y(l_{i-1}, l_i)$ zu l_{i-1}. Bei Erreichen des Literaturverzeichnisses berechnen wir den durchschnittlichen Zeilenabstand D_y, indem wir das arithmetische Mittel über alle gesammelten Zeilenabstände bilden. Damit die Berechnung von D_y nicht durch die üblicherweise größeren Zeilenabstände zu Überschriften und Kopf- oder Fußzeilen beeinflusst werden, berücksichtigen wir $d_y(l_{i-1}, l_i)$ nur dann, wenn der Zeilenabstand zwischen l_i und l_{i-1} äquivalent zum Zeilenabstand zwischen l_i und l_{i+1} ist, d.h. wenn $d_y(l_{i-1}, l_i) = d_y(l_i, l_{i+1})$. Unsere Annahme dabei ist, dass ein Zeilenabstand, der zweimal hintereinander auftritt, repräsentativ für den durchschnittlichen Zeilenabstand ist. Wir ignorieren mit dieser Bedingung die Zeilenabstände zwischen Zeilen, die sich in unterschiedlichen Spalten oder auf unterschiedlichen Seiten befinden. Weil Überschriften sowie Kopf- und Fußzeilen in der Regel aus nicht mehr als zwei Textzeilen bestehen, ignorieren wir zudem alle Zeilenabstände, die durch die Absetzung von Überschriften sowie Kopf- und Fußzeilen resultieren. Sollten dennoch zu große Zeilenabstände berücksichtigt werden, können wir diese durch die Bildung des arithmetischen Mittels relativieren: Sei m die Anzahl der Textzeilen l_i, für die die Bedingung $d_y(l_{i-1}, l_i) = d_y(l_i, l_{i+1})$ erfüllt wird. Der durchschnittliche Zeilenabstand D_y ergibt sich aus:

$$D_y = \frac{\sum\limits_{i,\, d_y(l_{i-1}, l_i) = d_y(l_i, l_{i+1})} d_y(l_{i-1}, l_i)}{m}$$

Umgang mit Kopf- und Fußzeilen Enthält die Publikation Kopf- und/oder Fußzeilen und erstreckt sich das Literaturverzeichnis zudem über mehrere Seiten, so werden diese Zeilen gemäß der Lesereihenfolge innerhalb der Referenzen „eingebettet“ (vgl. Abbildung 3.5). Da sie



(a) **(b)**

Abbildung 3.5: Beispiel einer wissenschaftlichen Publikation mit Kopf- und Fußzeilen und einem Seitenwechsel innerhalb des Literaturverzeichnisses. Entsprechend der Abarbeitungsreihenfolge der Textzeilen werden die Kopf- und Fußzeilen in das Literaturverzeichnis „eingebettet“.

aber nicht Teil des Literaturverzeichnisses sind und die Identifizierung der Referenzen beeinflussen, möchten wir derartige Zeilen herausfiltern.

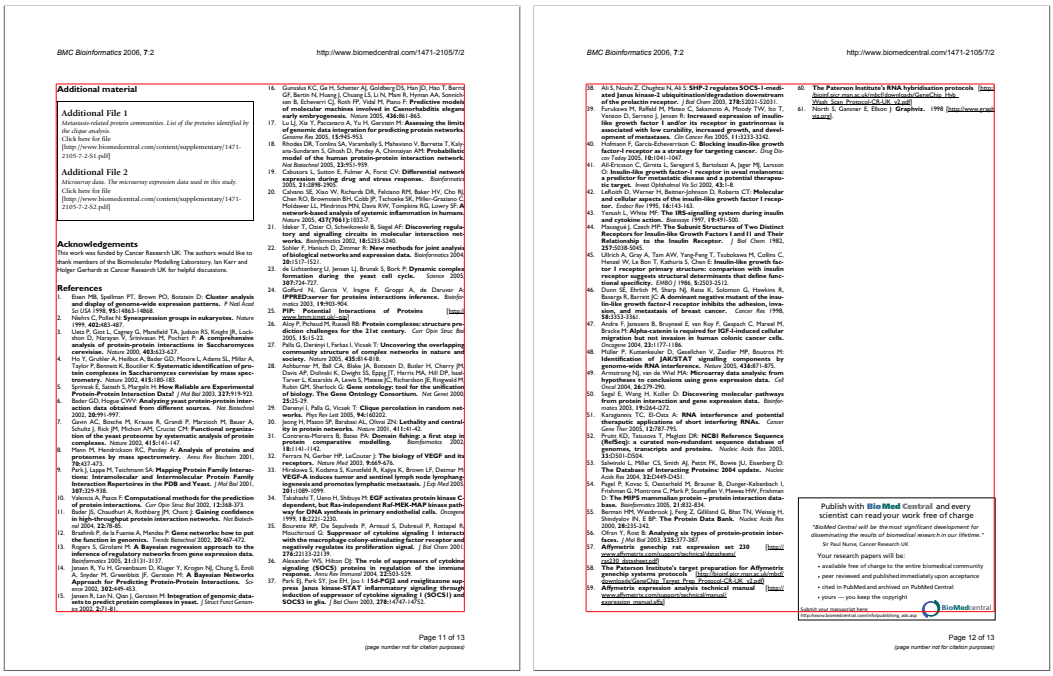
Unsere Idee hierzu ist die Definition einer Region *Content-Box*, die nur inhaltlich relevante Textzeilen (zu denen wir Kopf- und Fußzeilen nicht zählen) einer Seite enthält. Bei der Extraktion der Referenzen sollen nur Textzeilen, die Teil der Content-Box sind, berücksichtigt werden. Wir möchten mit der Content-Box im Folgenden also eine Region definieren, wie sie in Abbildung 3.6 dargestellt ist.

Seien $x(CB)$ und $y(CB)$ die X-Koordinaten der linken, oberen Ecke und $h(CB)$ die Höhe sowie $w(CB)$ die Breite der Content-Box. Wir werden die Content-Box über die Werte $x(CB)$, $y(CB)$, $h(CB)$ und $w(CB)$ eindeutig definieren.

Wir initialisieren $x(CB)$ und $y(CB)$ mit dem Wert $+\infty$ und $w(CB)$ sowie $h(CB)$ mit dem Wert $-\infty$ und passen die Werte immer dann an, wenn $d_y(l_{i-1}, l_i) = d_y(l_i, l_{i+1})$ für eine Textzeile l_i (mit $1 < i < n, n =$ Anzahl der Textzeilen bis zum Literaturverzeichnis) gilt. Hier gilt die gleiche Annahme wie für die Berechnung des durchschnittlichen Zeilenabstandes D_y: Die Kopf- und Fußzeilen sind meist durch einen größeren Zeilenabstand von inhaltlich relevanten Textzeilen abgesetzt und bestehen aus nicht mehr als zwei Textzeilen. Gleichzeitig zur Berechnung des durchschnittlichen Zeilenabstandes D_y überprüfen für jede Textzeile l_i, die diese Bedingung erfüllen, ob sie die Werte für die Content-Box verändern:

$$
\begin{aligned}
x(CB) &= \min(x(CB), x(l_{i-1})) \\
y(CB) &= \min(y(CB), y(l_{i-1})) \\
h(CB) &= \max(h(CB), y(l_{i+1}) + h(l_{i+1})) \\
w(CB) &= \max(w(CB), x(l_i) + w(l_i) - x(CB))
\end{aligned}
$$

Im Laufe der Verarbeitung der Textzeilen wird die X- und Y-Koordinate der Content-Box sukzessive verkleinert, während ihre Höhe und ihre Breite sukzessive erhöht wird. Üblicher-



(a) (b)

Abbildung 3.6: Beispiel einer Content-Box (rot markiert), die nur die inhaltlich relevanten Textzeilen der Publikation enthält. Mit der Content-Box können die Kopf- und Fußzeilen von der Extraktion ausgeschlossen werden.

weise wird $x(CB)$ und $y(CB)$ nur verändert, wenn l_{i-1} die erste Textzeile einer Spalte ist und $h(CB)$ nur dann verändert, wenn l_{i+1} die letzte Zeile einer Textspalte ist. Die Breite $w(CB)$ kann hingegen von beliebig vielen Textzeilen l_i verändert werden. Es ist nicht ausgeschlossen, dass eine inhaltlich relevante Textzeile l_{i-1}, l_i oder l_{i+1} eigentlich einen Wert von CB ändern würde, aber die Bedingung $d_y(l_{i-1}, l_i) = d_y(l_i, l_{i+1})$ nicht erfüllt wird, weshalb die Content-Box fälschlicherweise nicht angepasst wird. Wir nehmen aber an, dass die Grundstruktur für alle Seiten und Textspalten der Publikation gleich ist und dass bis zum Erreichen des Literaturverzeichnisses Textzeilen auftreten, die die Werte der Content-Box mindestens in gleichem Maße verändern und die geforderte Bedingung erfüllen. Der Fehler wäre somit egalisiert, so dass die Content-Box beim Erreichen der Überschrift des Literaturverzeichnisses die gewünschten Dimensionen besitzt.

3.3.2 Eine Heuristik zur Identifizierung von Referenzen

Sobald wir den Titel des Literaturverzeichnisses erreicht haben, identifizieren wir anhand der ihr folgenden Textzeilen, die Bestandteil der Content-Box sind, die Referenzen der Publikation. Wir versuchen dazu, für jede Textzeile l_i zu entscheiden, ob sie ein *Referenz-Titel* oder ein *Referenz-Anhang* ist.

Definition (Referenz-Titel & Referenz-Anhang)**.** *Eine Referenz besteht aus einem Referenz-Titel und möglicherweise aus einem Referenz-Anhang. Der Referenz-Titel bezeichne die erste Zeile der Referenz. Alle weiteren Zeilen der Referenz nennen wir Referenz-Anhang (vgl. Abbildung 3.7).*

Gelingt es uns, diese Entscheidung für jede Textzeile des Literaturverzeichnisses korrekt zu treffen, haben wir alle Referenzen des Literaturverzeichnisses identifiziert. Je nachdem, welche der folgenden Merkmale ein Literaturverzeichnis besitzt, ist diese Entscheidung aber unterschiedlich schwer zu treffen:

- **Referenz-Anker:** Kürzel, das dem Referenz-Titel zur eindeutigen Identifizierung vorangestellt ist. Üblich ist z.B. die Verwendung von aufeinander folgenden Zahlen (etwa [1],

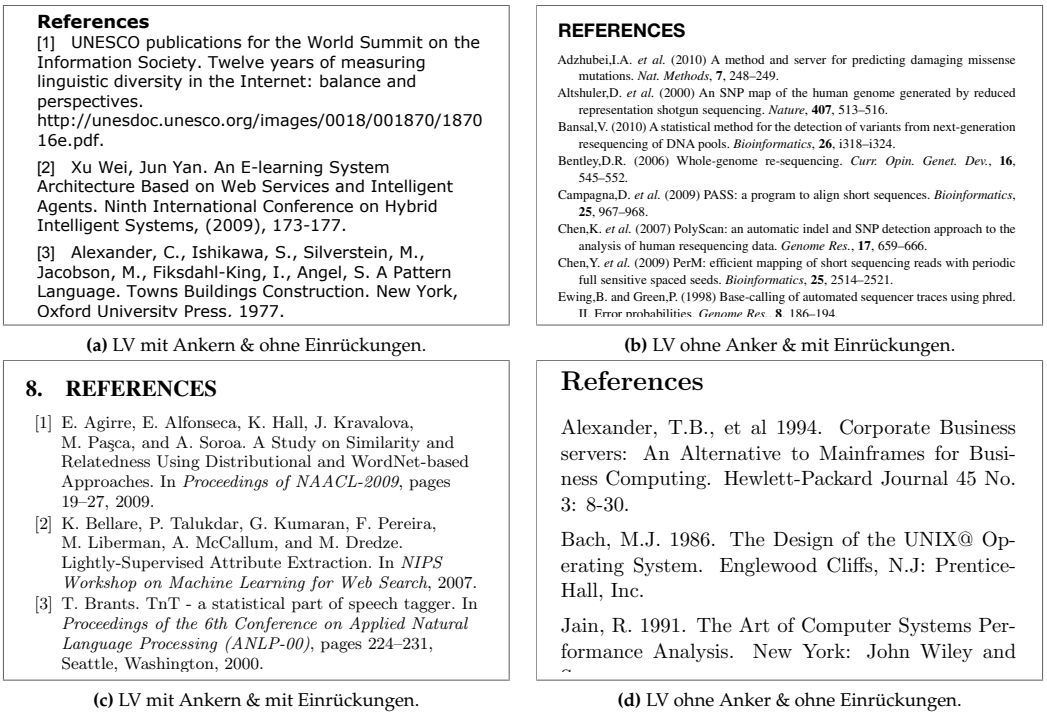
[1] E. Agirre, E. Alfonseca, K. Hall, J. Kravalova, M. Pasca, and A. Soroa. A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches. In Proceedings of NAACL-2009, pages 1927, 2009.

Abbildung 3.7: Beispiel einer Referenz mit Referenz-Titel (rot markiert) und Referenz-Anhang (blau markiert).

[2], [3], ... oder 1., 2., 3., ...) für die Durchnummerierung der Referenzen. Möglich sind auch Abkürzungen, die z.B. aus den Nachnamen der Autoren und dem Erscheinungsjahr zusammengesetzt sind (etwa [ChenR00], [Goodman03], [PietraPL95], etc.).

- **Einrückung des Referenz-Anhangs:** Der Referenz-Anhang ist gegenüber dem Referenz-Titel eingerückt, d.h. die X-Koordinaten der Textzeilen des Referenz-Anhangs sind im Vergleich zur X-Koordinate des Referenz-Titels größer.
- **Zeilenabstand zwischen Referenzen:** Der Zeilenabstand zwischen der letzten Zeile einer Referenz und dem Referenz-Titel der nachfolgenden Referenz ist größer als der Zeilenabstand zwischen den Textzeilen innerhalb einer Referenz.

Allgemein können alle Merkmale beliebig miteinander kombiniert sein. In Abbildung 3.8 sind einige Beispiele für verschiedenartig strukturierte Literaturverzeichnisse zu sehen. Abbildung



(a) LV mit Ankern & ohne Einrückungen.

(b) LV ohne Anker & mit Einrückungen.

(c) LV mit Ankern & mit Einrückungen.

(d) LV ohne Anker & ohne Einrückungen.

Abbildung 3.8: Beispiele für verschiedene Typen eines Literaturverzeichnisses (LV).

3.8a zeigt ein Literaturverzeichnis, in dem die Referenz-Titel Referenz-Anker enthalten, die Referenz-Anhänge jedoch nicht eingerückt sind. Im Gegensatz zu den Literaturverzeichnissen aus Abbildung 3.8b und 3.8c sind die Referenzen zudem durch einen größeren Y-Abstand voneinander abgetrennt. Abbildung 3.8b zeigt ein Literaturverzeichnis, bei dem die Referenz-Anhänge eingerückt sind, die Referenz-Titel aber keine Referenz-Anker besitzen. Das Literaturverzeichnis aus Abbildung 3.8c enthält sowohl Referenz-Titel mit Referenz-Anker als auch eingerückte Referenz-Anhänge und das Literaturverzeichnis aus Abbildung 3.8d keines der beiden Merkmale. Wir können folgende, allgemeingültige Beobachtungen festhalten:

Beobachtung 1. *Die erste Zeile eines Literaturverzeichnisses ist immer ein Referenz-Titel.*

Beobachtung 2. *Die Merkmale eines Literaturverzeichnisses sind konsistent, d.h. ein Referenz-Titel enthält genau dann einen Referenz-Anker, wenn alle Referenz-Titel einen Anker enthalten; ein Referenz-Anhang ist genau dann eingerückt, wenn alle Referenz-Anhänge eingerückt sind; alle Zeilenabstände zwischen den Referenzen sind gleich groß (mit Ausnahme bei Spalten- und Seitenwechseln innerhalb des Literaturverzeichnisses).*

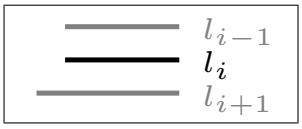
Beobachtung 3. *Referenz-Titel und Referenz-Anhänge besitzen ihre Merkmale exklusiv, d.h. Referenz-Titel sind nicht eingerückt und Referenz-Anhänge enthalten keine Anker.*

Beobachtung 4. *Das Einrückungslevel ist höchstens 1, d.h. es gibt keine Zeilen, die gegenüber eingerückten Zeilen nochmals eingerückt sind.*

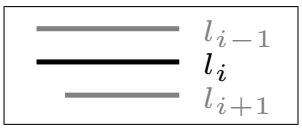
Zur Identifizierung der Referenzen entscheiden wir zunächst, welche der genannten Merkmale im Literaturverzeichnis vorliegen, anhand denen wir die Referenz-Titel von Referenz-Anhängen unterscheiden können. Konkret möchten wir wissen, ob die Referenz-Titel Referenz-Anker enthalten, und ob die Referenz-Anhänge eingerückt sind.

Mit Beobachtung 1 und 2 können wir bereits anhand der ersten Zeile des Literaturverzeichnisses entscheiden, ob die Referenz-Titel des Literaturverzeichnisses Referenz-Anker enthalten. Dazu überprüfen wir mit regulären Ausdrücken, ob diese Zeile einen Referenz-Anker enthält und schließen daraus, ob alle Referenz-Titel einen Anker haben. Ähnlich gehen wir vor, um festzustellen, ob Referenz-Anhänge eingerückt sind: Sobald wir eine Textzeile zum ersten Mal als Referenz-Anhang identifiziert haben, bestimmen wir, ob diese im Vergleich zum Referenz-Titel eingerückt ist und schließen daraus, ob alle Referenz-Anhänge eingerückt sind. Je nachdem, welche Merkmale wir vorfinden, können wir die Referenzen auf folgende Weise unterscheiden:

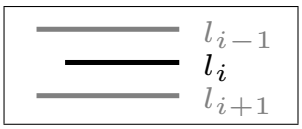
- **Literaturverzeichnis enthält Anker:** Wir benennen alle Textzeilen mit Ankern als Referenz-Titel und alle Textzeilen ohne Anker als Referenz-Anhänge.
- **Literaturverzeichnis enthält Einrückungen:** Um mögliche Einrückungen von Textzeilen identifizieren zu können, untersuchen wir jede Textzeile l_i zusammen mit ihrer vorherigen Textzeile l_{i-1} und ihrer nachfolgenden Textzeile l_{i+1}. Wegen Beobachtung 4 gibt es insgesamt sieben verschiedene Möglichkeiten, wie diese drei Textzeilen bezüglich ihrer X-Koordinaten zueinander angeordnet sein können. Entsprechend dieser Anordnungen können wir folgendermaßen entscheiden, ob l_i ein Referenz-Titel oder ein Referenz-Anhang ist:



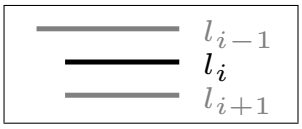
1. Die Textzeile l_{i-1} ist im Vergleich zu l_i nicht eingerückt, jedoch l_i im Vergleich zu l_{i+1}. Es handelt sich bei l_i um einen Referenz-Anhang, da es mit l_{i+1} eine Textzeile gibt, die l_i übergeordnet ist.



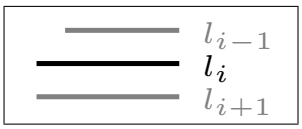
2. Die Textzeile l_{i+1} ist im Vergleich zu l_i eingerückt, l_{i-1} dagegen nicht. Es handelt sich bei l_i um einen Referenz-Titel, da l_i gegenüber l_{i+1} übergeordnet ist.



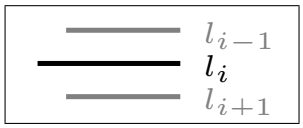
3. Die Textzeile l_i ist sowohl im Vergleich zu l_{i-1} als auch im Vergleich zu l_{i+1} eingerückt. Die Textzeile l_i ist also ein Referenz-Anhang.



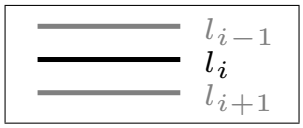
4. Die Textzeilen l_i und l_{i+1} sind im Vergleich zu l_{i-1} eingerückt, es handelt sich bei l_i also um einen Referenz-Anhang.



5. Die Textzeile l_{i-1} ist im Vergleich zu l_i eingerückt, es handelt sich bei l_i also um einen Referenz-Titel.



6. Sowohl l_{i-1} als auch l_{i+1} sind gegenüber l_i eingerückt. Die Textzeile l_i ist also ein Referenz-Titel.



7. Sowohl die vorherige Textzeile l_{i-1} als auch die nachfolgende Textzeile l_{i+1} ist im Vergleich zu l_i nicht eingerückt. Wir können also zumindest nicht anhand von Einrückungen eine Entscheidung treffen, ob es sich bei l_i um einen Referenz-Titel oder um einen Referenz-Anhang handelt. Wir haben aber angenommen, dass wir bereits wissen, dass die Referenz-Anhänge des Literaturverzeichnisses eingerückt sind. Da sich die Textzeilen l_{i-1} und l_i auf dem gleichen Einrückungslevel befinden und wir für l_{i-1} bereits eine Entscheidung getroffen haben, ist l_i genau dann ein Referenz-Titel, wenn l_{i-1} ein Referenz-Titel ist und genau dann ein Referenz-Anhang, wenn auch l_{i-1} ein Referenz-Anhang ist.

- **Literaturverzeichnis enthält (bisher) keine Einrückungen:** In diesem Fall besitzen alle drei betrachtete Textzeilen das gleiche Einrückungslevel (anderenfalls wäre mindestens eine Textzeile eingerückt und das Literaturverzeichnis würde somit Einrückungen enthalten). Zu beachten ist, dass das Literaturverzeichnis trotzdem eingerückte Referenz-Anhänge enthalten kann, wenn alle drei Textzeilen Referenz-Titel sind und wir im Laufe der Referenzen-Identifizierung noch keine Textzeile als Referenz-Anhang identifiziert haben.

  Wir identifizieren den Typ der Textzeile l_i über den Zeilenabstand zwischen den drei Textzeilen l_{i-1}, l_i und l_{i+1}: Sei $d_y(l_{i-1}, l_i)$ der Zeilenabstand zwischen der Textzeile l_{i-1} und l_i, und $d_y(l_i, l_{i+1})$ der Zeilenabstand zwischen der Textzeile l_i und l_{i+1}.

  Wenn $d_y(l_{i-1}, l_i) > d_y(l_i, l_{i+1})$, dann gehen wir davon aus, dass l_i ein Referenz-Titel ist, da l_i durch den größeren Abstand von der vorherigen Zeile abgesetzt ist und deshalb eine neue Referenz einleitet. Ein ähnliches Argument gilt, wenn $d_y(l_{i-1}, l_i) < d_y(l_i, l_{i+1})$: Wir gehen davon aus, dass l_i ein Referenz-Anhang ist, da der größere Abstand zu l_{i+1} darauf hinweist, dass mit l_{i+1} eine neue Referenz eingeleitet wird.

  Gilt hingegen $d_y(l_{i-1}, l_i) = d_y(l_i, l_{i+1})$, so beziehen wir den zuvor berechneten durchschnittlichen Zeilenabstand D_y mit ein: Wir vermuten, dass l_i genau dann ein Referenz-Titel ist, wenn $d_y(l_{i-1}, l_i) > D_y$. Ansonsten ist l_i ein Referenz-Anhang.

Umgang mit Seiten- / Spaltenwechseln in Literaturverzeichnissen Alle Entscheidungen haben wir bisher anhand der X- und Y-Koordinaten der Textzeilen getroffen. Dazu haben wir die Werte einer gerade betrachteten Textzeile mit den Werten der jeweils vorherigen oder der nachfolgenden Textzeile verglichen. Im Falle eines Spalten- oder Seitenwechsels, also wenn sich l_{i-1} oder l_{i+1} nicht in derselben Spalte oder auf derselben Seite wie l_i befindet, ist das aber nicht ohne weiteres möglich, weil die X-Koordinaten und die Y-Koordinaten der Textzeilen zu sehr voneinander abweichen und für unsere Zwecke nicht vergleichbar sind.

Nehmen wir zunächst an, dass der Spalten- bzw. Seitenwechsel zwischen den Textzeilen l_{i-1} und l_i stattfindet (unsere Ausführungen gelten genauso für den Fall, dass der Wechsel zwischen l_i und l_{i+1} stattfindet). Die Textzeile l_i ist also die erste Zeile einer Spalte (Seite). Wir möchten wissen, ob l_i ein Referenz-Titel oder ein Referenz-Anhang ist. Für diese Entscheidung unterscheiden wir wieder, welche Merkmale im Literaturverzeichnis vorliegen:

- **Literaturverzeichnis enthält Anker:** Dieser Fall bedarf keiner besonderen Behandlung: Die Textzeile l_i ist genau dann ein Referenz-Titel, wenn sie einen Referenz-Anker enthält, ansonsten ist l_i ein Referenz-Anhang.

- **Literaturverzeichnis enthält Einrückungen:** Wir können l_i als Referenz-Titel identifizieren, wenn l_{i+1} im Vergleich zu l_i eingerückt ist und umgekehrt als Referenz-Anhang, wenn l_i im Vergleich zu l_{i+1} eingerückt ist. Besitzen l_i und l_{i+1} hingegen das gleiche Einrückungslevel, können wir zunächst keine Entscheidung über l_i treffen, da wir über einen

Vergleich mit l_{i+1} keine Einrückung feststellen können und l_{i-1} für einen Vergleich nicht in Frage kommt, da die X-Koordinate von l_i im Vergleich zur X-Koordinate von l_{i-1} um einen für uns bisher unbekannten Wert α verschoben ist, so dass wir nicht entscheiden können, ob l_i im Vergleich zu l_{i-1} eingerückt ist. Unser Ansatz zur Lösung dieses Problems ist die Berechnung von α, um beide Textzeilen vergleichbar zu machen.

Sei k die Anzahl der Textspalten pro Seite der Publikation und CB die Content-Box (s.o.). Wir unterteilen CB vertikal in k gleich breite Bereiche CB_j, so dass jeder Bereich CB_j mit $1 \leq j \leq k$ genau die j-te Textspalte einer Seite enthält. Weiter sei $col(l_i)$ die Spaltennummer, in der sich l_i befindet und $w(CB)$ die Breite von CB. Es gilt:

$$\alpha = (col(l_i) - col(l_{i-1})) * \frac{w(CB)}{k} + \sigma$$

α ergibt sich, indem wir die Spalten zwischen l_{i-1} und l_i zählen und dementsprechend oft die Breite $w(CB_j) = \frac{w(CB)}{k}$ von CB_j zu α hinzufügen. Zusätzlich addieren wir einen Faktor σ hinzu, mit dem wir die Tatsache, dass eine Textspalte j einen linken und/oder rechten Innenabstand zu den Abgrenzungen von CB_j aufweisen kann, berücksichtigen (vgl. Abbildung 3.9). Die Idee zur Berechnung von σ ist folgende: Durch die Unterteilung von CB

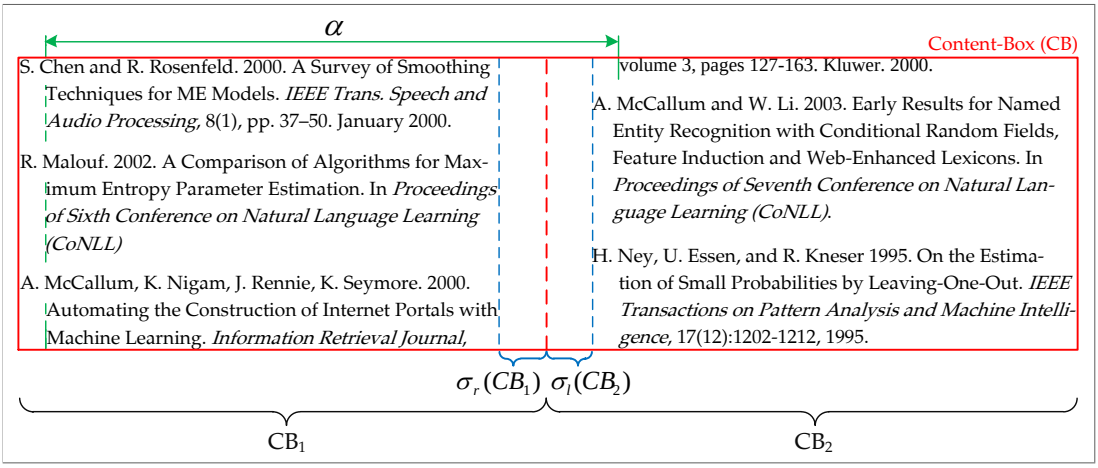


Abbildung 3.9: Veranschaulichung für die Berechnung des Wertes α, um zu entscheiden, ob die Textzeile *„volume 3, pages 127-163. Kluwer. 2000"* ein Referenz-Titel oder ein Referenz-Anhang ist.

in k gleich breite Bereiche haben wir den horizontalen Abstand zwischen der Textspalte $col(l_i) - 1$ und der Textspalte $col(l_i)$ in der Mitte geteilt (unter der Annahme, dass alle Textspalten gleich breit sind). Das bedeutet, dass der rechte Innenabstand $\sigma_r(CB_{col(l_i)-1})$ der Textspalte $col(l_i) - 1$ zur rechten Abgrenzung von $CB_{col(l_i)-1}$ (vgl. $\sigma_r(CB_1)$ in Abbildung 3.9) äquivalent zum linken Innenabstand $\sigma_l(CB_{col(l_i)})$ der Textspalte j zur linken Abgrenzung von $CB_{col(l_i)}$ ist (vgl. $\sigma_l(CB_2)$ in Abbildung 3.9).

Da wir zum Zeitpunkt der Bearbeitung von l_i bereits alle Textzeilen aus der Textspalte $col(l_i) - 1$ kennen, können wir ihren minimalen, rechten Innenabstand $\sigma_r(CB_{col(l_i)-1})$ zur rechten Abgrenzung $x_r(CB_{col(l_i)-1})$ von $CB_{col(l_i)-1}$ berechnen. Für $col(l_i) > 1$ gilt:

$$\sigma_l(CB_{col(l_i)}) = \sigma_r(CB_{col(l_i)-1}) = \min_{l_k \in CB_{col(l_i)-1}} (x_r(CB_{col(l_i)-1}) - x_r(l_k))$$

und

$$\sigma = \begin{cases} \sigma_r(CB_{col(l_i)-1}) & \text{, wenn } col(l_i) > col(l_{i-1}) \\ -\sigma_r(CB_{col(l_i)-1}) & \text{, wenn } col(l_i) < col(l_{i-1}) \\ 0 & \text{, sonst} \end{cases}$$

Zu beachten ist, dass die Berechnungen auch im Falle eines Seitenwechsels innerhalb eines Literaturverzeichnisses gelten, da bei einem Seitenwechsel ebenfalls eine neue Textspalte begonnen wird. Wir können einen Seitenwechsel deshalb als einen Spezialfall des

Spaltenwechsels interpretieren. Für den Fall, dass die Publikation mehr als eine Textspalte pro Seite enthält, ist bei einem Seitenwechsel $col(l_{i-1}) > col(l_i)$, weshalb sowohl σ als auch α negative Werte annehmen. Dies ist auch plausibel, da l_i im Vergleich zu l_{i-1} um einen negativen Wert verschoben ist. Enthält die Publikation hingegen nur eine Textspalte, so gilt bei einem Seitenwechsel: $col(l_{i-1}) = col(l_i)$, weshalb $\sigma = 0$ und $\alpha = 0$, da l_i im Vergleich zu l_{i-1} nicht verschoben ist.

Um zu entscheiden, ob l_i ein Referenz-Titel oder ein Referenz-Anhang ist, vergleichen wir nun $x(l_{i-1}) + \alpha$ mit $x(l_i)$:

$x(l_{i-1}) + \alpha > x(l_i)$: Die Textzeile l_{i-1} ist im Vergleich zu l_i eingerückt. Wenn l_{i-1} ein Referenz-Anhang ist, ist l_i also ein Referenz-Titel. Da das Einrückungslevel höchstens 1 ist, kann l_{i-1} theoretisch kein Referenz-Titel sein. Aufgrund möglicher Ungenauigkeiten der Daten können wir diesen Fall aber nicht ausschließen und identifizieren l_i gegebenenfalls ebenfalls als Referenz-Titel.

$x(l_{i-1}) + \alpha = x(l_i)$: Beide Textzeilen l_{i-1} und l_i befinden sich auf dem gleichen Einrückungslevel. Deshalb ist l_i genau dann ein Referenz-Titel, wenn l_{i-1} ebenfalls ein Referenz-Titel ist und genau dann ein Referenz-Anhang, wenn l_{i-1} ein Referenz-Anhang ist.

$x(l_{i-1}) + \alpha < x(l_i)$: Die Textzeile l_i ist im Vergleich zu l_{i-1} eingerückt. Wenn l_{i-1} ein Referenz-Titel ist, ist l_i ein Referenz-Anhang. Wegen der gleichen Argumentation des 1. Falls, kann l_{i-1} theoretisch kein Referenz-Anhang sein. Ist dies trotzdem der Fall, identifizieren wir l_i ebenfalls als Referenz-Anhang.

- **Literaturverzeichnis enthält keine Einrückungen:** Wenn das Literaturverzeichnis weder Referenz-Anker noch eingerückte Referenz-Anhänge enthält, ist die Entscheidung, ob die erste Textzeile einer Spalte ein Referenz-Titel oder ein Referenz-Anhang ist, über die Untersuchung der Positionen von Textzeilen nur schwer zu lösen (zur Verdeutlichung dieses Szenarios stelle man sich das Literaturverzeichnis aus Abbildung 3.9 ohne Einrückungen vor). Das sich die Textzeilen l_{i-1} und l_i in unterschiedlichen Spalten bzw. auf unterschiedlichen Seiten befinden, ist der Zeilenabstand zwischen beiden Textzeilen für unsere Zwecke aus offensichtlichen Gründen nicht verwendbar.

  Wir suchen deshalb in der *Semantik* einer Textzeile nach Kriterien, mit denen wir den Typ von l_i bestimmen können. Wir identifizieren l_i als Referenz-Anhang, wenn:

  - l_{i-1} mit dem Zeichen `","`, `";"` oder `":"` endet. Alle aufgeführten Zeichen implizieren, dass noch Inhalt folgt. Wir können also davon ausgehen, dass die Textzeilen l_{i-1} und l_i in einem Kontext stehen und l_i deshalb ein Referenz-Anhang ist.
  - l_i mit einem Kleinbuchstaben beginnt. Da Referenz-Titel die erste Zeile der Referenz darstellen, beginnen diese üblicherweise mit einem Großbuchstaben[1]. Wir gehen deshalb davon aus, dass die Textzeile l_i kein Referenz-Titel sein kann, wenn sie mit einem Kleinbuchstaben beginnt.

Identifizieren des Endes eines Literaturverzeichnisses Bisher haben wir die Referenzen so untersucht, als ob alle Textzeilen nach der Überschrift des Literaturverzeichnisses auch tatsächlich zum Literaturverzeichnis gehören. Wir haben aber bereits zu Beginn des Kapitels erwähnt, dass jedoch noch andere Abschnitte (z.B. Abbildungs- oder Tabellenverzeichnisse) folgen können, die für die Referenzen-Extraktion nicht relevant sind und wir deshalb nicht extrahieren wollen. Für jede Textzeile müssen wir deshalb zusätzlich überprüfen, ob sie noch Teil des Literaturverzeichnisses ist.

Wir suchen also Kriterien, anhand derer wir feststellen können, ob eine Textzeile l_i nicht mehr relevant für die Referenzen-Extraktion ist. Ein naheliegendes Kriterium ist die Schriftauszeichnung von l_i: Hebt sich die Textzeile l_i durch ihre Schriftgröße, ihre Schriftart oder ihren Schriftschnitt von den vorherigen Textzeilen des Literaturverzeichnisses ab, können wir

[1] Eine Ausnahme bilden hier Autorennamen mit Namenszusätzen wie z.B. „von“, „van“ oder „zu“. Wird ein derartiger Name in einer Referenz zuerst aufgelistet, kann ein Referenz-Titel auch mit einem Kleinbuchstaben beginnen.

davon ausgehen, dass l_{i-1} die letzte Textzeile des Literaturverzeichnisses war und wir somit l_i nicht mehr berücksichtigen müssen. Weiterhin können wir Textzeilen ausschließen, wenn sie die Merkmals

Ein weiteres Kriterium ist der Zeilenabstand von l_i zu l_{i-1}. Wenn dieser Zeilenabstand im Vergleich zum durchschnittlichen Zeilenabstand D_y zu groß ist, gehört l_i nicht mehr zum Literaturverzeichnis. Allerdings ist dieses Kriterium nicht unproblematisch, weil wir anhand des Zeilenabstandes bereits die Referenzen unterscheiden, wenn im Literaturverzeichnis keine Anker und keine Einrückungen existieren. Insbesondere, wenn der Zeilenabstand größer als D_y ist, müssen wir entscheiden, ob die Textzeile den Beginn einer neuen Referenz oder das Ende des Literaturverzeichnisses darstellt. Mithilfe einer Toleranz von z.B. $2 * D_y$, in der der Zeilenabstand zwischen l_{i-1} und l_i liegen muss, damit l_i noch als Teil des Literaturverzeichnisses erkannt wird, können wir zwar versuchen, beide Fälle zu unterscheiden – dennoch bleibt fraglich, wie genau wir damit jeweils das Ende eines Literaturverzeichnisses erkennen können.

Unter anderem diese Tatsache macht deutlich, dass wir die Referenzen wie die Titel von wissenschaftlichen Publikationen lediglich mit einer Heuristik identifizieren. Wir müssen also damit rechnen, dass nicht alle Referenzen korrekt extrahiert werden. Um festzustellen, mit welcher Genauigkeit die Referenzen identifiziert werden, evaluieren wir in unseren Experimenten deshalb auch die Referenzen-Extraktion (vgl. Kapitel 3.3).

Mithilfe einer geeigneten Fehlertoleranz bei der nun folgenden Titel- und Referenzen-Zuordnung sind Fehler bei der Extraktion aber solange vertretbar, bis der repräsentierende Eintrag in einer Literaturdatenbank immer noch gefunden wird. Diesem können wir dann die korrekte und vollständigen Metadaten einer Publikation entnehmen.

Kapitel 4

Zuordnung von Titel & Referenzen

Um aus den extrahierten Informationen die korrekten Metadaten zu erhalten und ihre Felder eindeutig identifizieren zu können, wollen wir nun jeder im vorigen Kapitel extrahierten Zeichenkette einem Eintrag einer Literaturdatenbank zuordnen, sofern ein solcher Eintrag existiert. Die Schwierigkeit dieser Zuordnung besteht darin, dass wir sowohl bei der Zeichenextraktion als auch bei der Titel- bzw. Referenzen-Identifizierung Fehler nicht ausschließen können und deshalb die Informationen aus einer extrahierten Zeichenkette nicht vollständig mit den Informationen des repräsentierenden Eintrags der Literaturdatenbank übereinstimmen müssen. Die Zuordnung muss also fehlertolerant sein, damit trotz möglicher Fehler in der extrahierten Zeichenkette der repräsentierende Eintrag in der Literaturdatenbank gefunden wird. Allerdings soll die vorhandene Fehlertoleranz nicht dazu führen, dass einer extrahierten Zeichenkette ein falscher Eintrag zugeordnet wird.

Dazu werden wir zunächst über eine indexbasierte Suche eine Auswahl möglicher Zuordnungs-Kandidaten bestimmen, die wir danach jeweils nach der Ähnlichkeit zur extrahierten Zeichenkette bewerten. Wenn seine Bewertung einen definierten Schwellenwert erreicht, ist derjenige Kandidat mit der höchsten Bewertung der gesuchte Eintrag der Literaturdatenbank.

4.1 Literaturdatenbanken

Bevor wir damit beginnen, die Zuordnung von Titeln und Referenzen wissenschaftlicher Publikationen zu Einträgen von Literaturdatenbanken zu erläutern, sei zunächst der Begriff der Literaturdatenbank definiert:

Definition (Literaturdatenbank)**.** *Eine Literaturdatenbank speichert zu wissenschaftlichen Publikationen eines bestimmten Fachgebietes ihre bibliografischen Metadaten, wie z.B. den Titel, die Autoren, das Erscheinungsjahr oder den Herausgeber.*

Zwei Beispiele für Literaturdatenbanken, die im weiteren Verlauf Anwendung finden werden, sind **DBLP** und **Medline**:

- **DBLP:** DBLP ist eine Literaturdatenbank aus dem Bereich der Informatik, die mehr als 1,7 Millionen Einträge ([42], Stand: September 2011) enthält. Alle Einträge werden in einer etwa 850MB großen XML-Datei namens `dblp.xml` gespeichert, von der in Beispiel 4.1 ein Ausschnitt zu sehen ist.

  Jeder Eintrag repräsentiert eine wissenschaftliche Publikation und enthält folgende Felder:

  - `mdate:` Datum, an dem der Eintrag zuletzt geändert wurde.
  - `key:` Der Schlüssel, mit dem wir jeden Eintrag eindeutig identifizieren können.
  - `author:` Ein Autor der Publikation. Alle Autoren einer Publikation werden mit Vor- und Nachnamen in einem eigenen `author`-Element gelistet.
  - `title:` Der Titel der Publikation.

```
<dblp>
   [...]
   <inproceedings mdate="2002-12-09" key="conf/sigcomm/KrishnamurthyW00">
      <author>Balachander Krishnamurthy</author>
      <author>Jia Wang</author>
      <title>On network-aware clustering of web clients.</title>
      <pages>97-110</pages>
      <year>2000</year>
      <booktitle>SIGCOMM</booktitle>
      <ee>http://doi.acm.org/10.1145/347059.347412</ee>
      <url>db/conf/sigcomm/sigcomm2000.html#KrishnamurthyW00</url>
   </inproceedings>
   [...]
</dblp>
```

Beispiel 4.1: Ausschnitt aus `dblp.xml`. Der dargestellte Datensatz enthält die Attribute `mdate` und `key` sowie die Felder `author`, `title`, `pages`, `year`, `booktitle`, `ee` und `url`.

  - `pages`: Position, an der die Publikation innerhalb des Konferenzbandes zu finden ist.
  - `year`: Das Erscheinungsjahr der Publikation.
  - `booktitle`: Der Titel des Konferenzbandes.
  - `ee`: URL zur Webseite, auf der der Verleger die Publikation üblicherweise als digitales Dokument (z.B. im PDF-Format) verbreitet.
  - `url`: URL zu einer individuellen Webseite von DBLP, auf der weitere Informationen zur Publikation zu finden sind.

- **Medline** Literaturdatenbank für Publikationen aus dem Bereich der Medizin, die knapp 19 Millionen Einträge – aufgeteilt auf über 650 XML-Dateien – ab dem Jahr 1966 enthält ([23], Stand: August 2011). Ein Medline-Eintrag enthält üblicherweise mehr Metadaten als ein DBLP-Eintrag, wie z.B. eine Zusammenfassung (*Abstract*) der Publikation, das Erstellungsjahr[1] oder die Sprache, in der die Publikation verfasst wurde (vgl. Beispiel 4.2). Um beide Literaturdatenbanken einheitlich verarbeiten zu können, erstellen wir aus den 650 XML-Dateien eine zu der Struktur von `dblp.xml` äquivalente XML-Datei `medline.xml`, für die wir folgende für uns relevanten Metadaten (das sind diejenigen Werte, für die vergleichbare Werte in `dblp.xml` existieren), übernehmen:
  - `key`: Schlüssel, mit dem wir jeden Medline-Eintrag eindeutig identifizieren können. Im Gegensatz zu einem DBLP-Key ist der Medline-Key numerisch.
  - `author`: Ein Autor der Publikation. Alle Autoren einer Publikation werden mit Vor- und Nachnamen in einem eigenen `author`-Element gelistet.
  - `title`: Der Titel der Publikation.
  - `year`: Das Erscheinungsjahr der Publikation.

  Wir erhalten eine etwa 5,3GB große XML-Datei `medline.xml`, von der in Beispiel 4.3 ein Ausschnitt mit einem Medline-Eintrag zu sehen ist.

4.2 Allgemeines Prinzip der Zuordnung

Sowohl die Titel- als auch die Referenzen-Zuordnung lässt sich durch folgende Problemstellung beschreiben: Gegeben sei eine Suchanfrage Q, die entweder aus einem extrahierten Titel oder einer extrahierten Referenz besteht. Gesucht ist derjenige Eintrag E_Q aus der Literaturdatenbank DB_L (DBLP oder Medline), der die in Q beschriebene Publikation repräsentiert. Falls Q keine wissenschaftliche Publikation beschreibt oder falls für Q kein entsprechender Eintrag E_Q in DB_L existiert, so soll Q auch tatsächlich keinem Eintrag E_Q zugeordnet werden.

Zur Lösung dieses Problems werden wir zunächst mithilfe einer indexbasierten Suche eine Auswahl C möglicher Kandidaten aus DB_L ermitteln, aus denen wir in einem zweiten Schritt

[1] Das Erstellungsjahr einer Publikation ist nicht zu verwechseln mit ihrem Erscheinungsjahr.

```
<MedlineCitationSet>
   [...]
   <MedlineCitation Owner="NLM" Status="PubMed-not-MEDLINE">
      <PMID Version="1">19867238</PMID>
      <DateCreated>
         <Year>2010</Year>
         <Month>06</Month>
         <Day>22</Day>
      </DateCreated>
      <DateCompleted>
         <Year>2010</Year>
         <Month>06</Month>
         <Day>22</Day>
      </DateCompleted>
      <Article PubModel="Print">
         <Journal>
            <ISSN IssnType="Print">0022-1007</ISSN>
            <JournalIssue CitedMedium="Print">
               <Volume>11</Volume>
               <Issue>1</Issue>
               <PubDate>
                  <Year>1909</Year>
                  <Month>Jan</Month>
                  <Day>9</Day>
               </PubDate>
            </JournalIssue>
            <Title>The Journal of experimental medicine</Title>
            <ISOAbbreviation>J. Exp. Med.</ISOAbbreviation>
         </Journal>
         <ArticleTitle>ON THE RELATION OF TETANY TO T[...]</ArticleTitle>
         <Pagination>
            <MedlinePgn>118-51</MedlinePgn>
         </Pagination>
         <Abstract>
            <AbstractText>
               1. Tetany occurs spontaneously in many forms and may [...]
            </AbstractText>
         </Abstract>
         <Affiliation>Hunterian Laboratory and the Med[...]</Affiliation>
         <AuthorList CompleteYN="Y">
            <Author ValidYN="Y">
               <LastName>Maccallum</LastName>
               <ForeName>W G</ForeName>
               <Initials>WG</Initials>
            </Author>
            <Author ValidYN="Y">
               <LastName>Voegtlin</LastName>
               <ForeName>C</ForeName>
               <Initials>C</Initials>
            </Author>
        </AuthorList>
        <Language>eng</Language>
      </Article>
   </MedlineCitation>
   [...]
</MedlineCitationSet>
```

Beispiel 4.2: Die ursprüngliche Struktur eines Eintrags aus Medline, der mit `PMID`, `ArticleTitle`, `Author` und `DateCompleted` vergleichbare Felder mit einem DBLP-Eintrag hat, aber auch z.B. mit `DateCreated`, `Abstract`, `Language` individuelle Felder besitzt.

```
<medline>
   [...]
   <article key="19867238">
      <author>W G Maccallum</author>
      <author>C Voegtlin</author>
      <title>ON THE RELATION OF TETANY TO THE PARATHYROID GL[...]</title>
      <year>2010</year>
   </article>
   [...]
</medline>
```

Beispiel 4.3: Ausschnitt aus `medline.xml`. Der dargestellte Eintrag enthält mit `key`, `author`, `title` und `year` alle Felder aus Medline, zu denen vergleichbare Felder in DBLP existieren.

denjenigen Kandidaten $C_i \in C$ als E_Q bestimmen, der Q am ähnlichsten ist. Alle im Folgenden beschriebenen Datenstrukturen und Algorithmen wurden in der Programmiersprache *C++* implementiert.

4.2.1 Das Prinzip eines invertierten Index

Die Grundlage unserer indexbasierten Suche ist ein *invertierter Index*.

Definition (Invertierter Index)**.** *Sei n die Anzahl der Wörter in DB_L und E die Menge von Einträgen in DB_L. Ein invertierter Index ist eine Datenstruktur, die jedes Wort t_i $(1 \leq i \leq n)$ auf eine Liste (auch invertierte Liste genannt) abbildet, die alle Einträge $E_j \in E$ speichert, die t_i enthalten. Jeder Eintrag E_j wird dabei durch eine eindeutige, numerische ID j repräsentiert.*

Folgendes Beispiel zeigt einen invertierten Index, in dem das Wort t_1 u.a. in den Einträgen E_1, E_4, E_5 und E_{12}, das Wort t_2 u.a. in den Einträgen E_2, E_3 und E_4 sowie das Wort t_3 u.a. in den Einträgen E_4, E_7, E_8 enthalten ist:

$$\begin{aligned} t_1 &\mapsto 1, 4, 5, 12, \ldots \\ t_2 &\mapsto 2, 3, 4, \ldots \\ t_3 &\mapsto 4, 7, 8, \ldots \\ &\ldots \\ t_n &\mapsto \ldots \end{aligned}$$

Grundsätzlich ist das Prinzip eines invertierten Index mit dem eines Stichwortverzeichnisses in einem Buch zu vergleichen, da beide Modelle eine schnelle Identifizierung aller Positionen, an denen ein bestimmtes Wort auftritt, ermöglichen. Mithilfe eines invertierten Index, der über alle Wörter aus DB_L erstellt wurde, wird es uns also möglich sein, alle Einträge, die ein gegebenes Wort in ihren Metadaten enthalten, schnell zu finden.

Wir realisieren den invertierten Index mit einer *Hashmap*, mit der wir ein Wort t_i auf ihre invertierte Liste abbilden. Bei einer Hashmap wir der Speicherort eines Elements (in unserem Fall eine invertierte Liste) durch einen eindeutigen Schlüssel (in unserem Fall ein Wort t_i) und einer geeigneten Hashfunktion festgelegt, wodurch das Einfügen und Suchen eines Elements in durchschnittlich konstanter Zeit möglich ist. Beim Einfügen eines Elements treten Kollisionen auf, wenn die Hashfunktion mindestens zwei unterschiedliche Elemente dem gleichen Speicherort zuordnet. Solche Kollisionen treten insbesondere dann auf, wenn die Anzahl der einzufügenden Elemente die Anzahl verfügbarer Speicherorte übersteigt. Je nach Implementierung der Hashmap werden Kollisionen auf unterschiedliche Weise behandelt. So werden bei einer Hashmap mit Verkettung alle Elemente an ihrem zugeordneten Speicherplatz z.B. in einer Liste gespeichert. Im Falle einer Kollision an einem Speicherort wird das Element der entsprechenden Liste angehängt.

Um ein bestimmtes Element zu suchen, berechnen wir zunächst über einen eindeutigen Schlüssel und der Hashfunktion seinen festgelegten Speicherort und überprüfen, ob sich dort das gesuchte Element befindet. Die Laufzeit der Suche hängt dabei von der Anzahl der Kollisionen an diesem Speicherort ab: Gibt es keine Kollisionen, so ist die Suche nach einem Element

in Zeit $O(1)$ zu bewerkstelligen. Treten am Speicherort hingegen Kollisionen auf, müssen wir die dort vorhandene Liste durchsuchen. Im ungünstigsten Fall (nämlich wenn alle Elemente dem gleichen Speicherort zugeordnet wurden) benötigt die Suche Zeit in $O(n)$, weil die Listen üblicherweise nicht sortiert sind.

Der Ansatz des *Linearen Hashings* möchte diesem Fall entgegenwirken, indem die Anzahl der verfügbaren Speicherorte sukzessive erhöht wird, wenn der Belegungsfaktor einer Liste zu groß wird. Dadurch gelingt es, die Laufzeit für das Einfügen und Suchen eines Element im Durchschnitt auf $O(1)$ zu halten (vgl. dazu zum Beispiel [47]).

4.2.2 Die Erstellung eines invertierten Index

Zur Erstellung des invertierten Index parsen wir die XML-Dateien `dblp.xml` und `medline.xml` und entnehmen jedem ihrer Einträge lediglich den Key, den Titel, die Autoren und das Erscheinungsjahr. Alle anderen Metadaten sind für die Zuordnung irrelevant und werden aus Gründen der Performanz nicht mit in den Index aufgenommen. In Kapitel 6.2 werden wir erklären, warum uns für die Benutzerschnittstelle trotzdem die vollständigen Metadaten zur Verfügung stehen werden. Mit den ausgelesenen Daten erstellen wir den invertierten Index folgendermaßen:

1. **Normalisiere Metadaten und zerlege sie in ihre Einzelwörter:**

   (a) Wandle alle Buchstaben in Kleinbuchstaben um:

   ```
   Blakeley , 1995 , OQL[C++]: Extending C++ with an Object Query Capability.
                                        ⇓
   blakeley 1995 oql[c++]: extending c++ with an object query capability.
   ```

   (b) Teile Metadaten an jedem Zeichen auf, das kein Buchstabe und keine Zahl ist:

   ```
   blakeley 1995 oql[c++]: extending c++ with an object query capability.
                                        ⇓
   {blakeley, 1995, oql, c, extending, c, with, an, object, query, capability}
   ```

   (c) Entferne alle Stoppwörter[2] und alle Wörter, deren Länge kleiner als drei ist:

   ```
   {blakeley, 1995, oql, c, extending, c, with, an, object, query, capability}
                                        ⇓
   {blakeley, 1995, oql, extending, object, query, capability}
   ```

2. **Füge Einzelwörter in invertierten Index ein:**
   Jedes Einzelwort fügen wir zusammen mit einer eindeutigen ID j, die sich aus einer fortlaufenden Nummerierung der Einträge ergibt, in den als Hashmap implementierten Index ein. Falls für das Wort bereits eine invertierte Liste in der Hashmap existiert, erweitern wir die Liste um die ID. Ansonsten erstellen wir eine neue Liste, fügen die ID in diese Liste ein und speichern die Liste an dem durch das Wort und die Hashfunktion bestimmten Speicherort innerhalb der Hashmap. Um später von der ID wieder auf den Key des Eintrags schließen zu können, speichern wir außerdem den Key $\texttt{key}(E_j)$ des Eintrags E_j der Reihenfolge nach in einem Vektor `keys` ab, so dass gilt:

   $$\texttt{keys[j]} = \texttt{key}(E_j)$$

4.2.3 Die Suche mit einem invertierten Index

Um den die Suchanfrage Q repräsentierenden Eintrag E_Q zu finden, suchen wir zunächst mithilfe des gerade erstellten invertierten Index alle zu Q relevanten Einträge in DB_L. Dazu normalisieren wir Q auf die gleiche Weise wie oben beschrieben (Umwandlung in Kleinbuchstaben, Aufteilung in Wörter, Entfernen der Stoppwörter). Für jedes dabei resultierende Wort holen wir

Suchanfrage Q:	`Extending C++ with an Object Query Capability.`
Q, normalisiert:	{`extending`, `object`, `query`, `capability`}

invertierte Listen:	`extending` $\mapsto$ `2, 5, 7`
	`object` $\mapsto$ `1, 2, 4, 5`
	`query` $\mapsto$ `5`
	`capability` $\mapsto$ `2, 4, 5`

Beispiel 4.4: Ablauf einer Suche mit einem invertierten Index. Die Suchanfrage *„Extending C++ with an Object Query Capability."* wird normalisiert und für jedes resultierende Wort die invertierte Liste aus dem Index abgefragt.

uns in durchschnittlich konstanter Zeit die entsprechende invertierte Liste aus dem invertierten Index:

Zur Erinnerung: Jede invertierte Liste eines Wortes t_i enthält die ID's aller Einträge E_j, die t_i entweder im Feld `title`, im Feld `author` oder im Feld `year` enthalten. Je öfter also eine ID in diesen Listen vorkommt, desto mehr Wörter hat der entsprechende Eintrag mit der Suchanfrage Q gemeinsam, desto relevanter ist er zur Suchanfrage Q und desto wahrscheinlicher ist es, dass der Eintrag der gesuchte Eintrag E_Q ist. Unser (temporäres) Ziel ist also die Berechnung einer Liste L_C, die alle ID's aus den invertierten Listen, die wir für Q aus dem Index geholt haben, absteigend sortiert nach ihren Häufigkeiten in den invertierten Listen, enthält. Konkret möchten wir für die invertierten Listen aus Beispiel 4.4 die Liste $L_C = \{5,2,4,1,7\}$ berechnen.

Unsere Aufgabe besteht also darin, k (mit $k > 1$) invertierte Listen auf eine geeignete Weise zu vereinigen und ihre Elemente nach ihren Häufigkeiten in den invertierten Listen zu sortieren, so dass die gewünschte Liste L_C resultiert. Dabei können wir uns die Tatsache zu Nutze machen, dass die invertierten Listen dank der sequentiellen Durchnummerierung der Literaturdatenbank-Einträge aufsteigend sortiert sind. Betrachten wir als Beispiel dazu zunächst den einfachsten Fall für $k = 2$:

Wir berechnen zunächst die Vereinigung $L_M = L_1 \cup L_2$ der beiden Listen Listen L_1 und L_2. Die Liste L_C resultiert dann aus der Sortierung der Elemente aus L_M nach ihren Häufigkeiten. Für die Vereinigung $L_M = L_1 \cup L_2$ betrachten wir das jeweils erste Element beider Listen, aus denen wir das Minimum bestimmen. Die aktuell betrachteten Elemente sind im folgenden Beispiel farbig unterlegt (wenn es sich um das Minimum handelt blau, sonst rot):

L_1: {2, 5, 7}
L_2: {1, 2, 4, 5}
L_M: {}

Weil die Listen sortiert sind und jedes Element in einer Liste nur einmal vorkommt, betrachten wir im Moment alle Vorkommen des Minimums (es ist also ausgeschlossen, dass dieses Minimum im weiteren Verlauf der Vereinigung erneut auftritt). Wir entfernen das Minimum id_{min} aus allen Listen, in denen es vorkommt und fügen es als ein Element (id_{min}, i) zusammen mit seiner Häufigkeit i mit $1 \leq i \leq k$ in die vereinigende Liste L_M ein.

L_1: {2, 5, 7}
L_2: {1, 2, 4, 5}
L_M: {(1,1)}

Dies tun wir solange, bis wir alle Elemente durchlaufen haben und die Vereinigung abgeschlossen ist. Es resultiert die vollständig vereinigte Liste L_M:

[2]Stoppwörter sind Wörter, die allgemein sehr häufig auftreten und deshalb irrelevant für eine aussagekräftige Suche sind. Eine Liste möglicher Stoppwörter ist in Beispiel B.6 zu finden.

$$L_M: \quad \texttt{\{(1,1); (2,2); (4,1); (5,2); (7,1)\}}$$

die wir anschließend nach den Häufigkeiten der Elemente sortieren, wobei wir alle Elemente mit der gleichen Häufigkeit aufsteigend nach ihrer ID sortieren. Wir erhalten für L_C folgende Liste:

$$L_C: \quad \texttt{\{(2,2); (5,2); (1,1); (4,1); (7,1)\}}$$

Zur Analyse der Laufzeit dieser Methode zur Vereinigung zweier invertierter Listen nehmen wir der Einfachheit halber an, dass beide Listen jeweils m Elemente enthalten. Da wir jedes Element genau einmal betrachten und für jedes Element nur Operationen in konstanter Zeit ausführen (Minimum aus zwei Elementen bestimmen, Minimum aus höchstens 2 Listen entfernen, Minimum in neue Liste einfügen), ergibt sich für die Vereinigung von zwei Listen eine Laufzeit von $O(2 * m) = O(m)$.

Um nun k Listen zu vereinigen, könnten wir als einen naiven Ansatz oben beschriebene Methode für $k = 2$ insgesamt $(k-1)$-mal verschachteln, indem wir die Vereinigung über $(...(L_1 \cup L_2) \cup L_3) \cup ...) \cup L_{k-1}) \cup L_k)$ berechnen. Im ungünstigsten Fall (nämlich wenn sich alle Elemente der Listen untereinander unterscheiden), müssten wir dann im ersten Vereinigungsschritt insgesamt $(2 * m)$-viele Elemente vereinigen, im zweiten Schritt $(3 * m)$-viele Elemente (die aus dem ersten Schritt resultierende Liste enthält $(2 * m)$-viele Elemente + m Elemente aus L_3), im dritten Schritt $(4 * m)$-viele Elemente, usw. Insgesamt müssten wir also

$$\begin{aligned} 2m + 3m + 4m + \ldots + km &= \sum_{i=2}^{k} i * m \\ &= m * (\sum_{i=1}^{k} i) - m \\ &= m * \frac{k(k+1)}{2} - m \end{aligned}$$

Berechnungsschritte ausführen, woraus eine Laufzeit von $O(k^2 * m)$ resultieren würde. Eine anschließende Sortierung nach der Häufigkeit der Elemente würde bei insgesamt $(k * m)$-vielen Elementen Zeit in $O(km \log km)$ in Anspruch nehmen. Es resultiert schließlich eine Gesamtlaufzeit von $O(k^2 m + km \log km) = O(k^2 m)$ für diesen naiven Ansatz.

Unser Ansatz zur Reduzierung dieser Laufzeit auf insgesamt $O(km \log km)$ setzt am Prinzip der Methode für $k = 2$ an:

```
1:    { 2 , 5, 7}
2:    { 1 , 2, 4, 5}
3:    { 5 }
4:    { 2 , 4, 5}
```

PQ_{occ}: {}

Äquivalent zum Fall für $k = 2$ betrachten wir zu Beginn alle jeweils ersten Elemente der Listen, aus denen wir das Minimum bestimmen. Alle aktuell betrachteten Elemente sind farbig unterlegt (wenn es sich um das Minimum handelt blau, sonst rot). Während wir die Berechnung des Minimums für $k = 2$ mit nur einem Vergleich bewerkstelligen konnten, benötigen wir für k Elemente $k - 1$ Vergleiche, was eine Laufzeit in $O(k)$ zur Folge hätte. Im Vergleich zur naiven Methode hätten wir dadurch keine Vorteile in der Laufzeit erreicht, da sich die Gesamtlaufzeit erneut auf $O(k^2 m)$ belaufen würde.

Stattdessen fügen wir die aktuell betrachteten Elemente der Listen in eine Priorityqueue PQ_{min} ein, in der die Elemente aufsteigend nach ihren Werten sortiert werden. Die Ausgabe des

Minimums ist mit einer Priorityqueue in konstanter Zeit und das Einfügen sowie das Löschen eines Elements in logarithmischer Zeit möglich. Da die invertierten Listen aufsteigend sortiert sind, können wir sicher gehen, dass alle Vorkommen dieses Minimums in PQ_{min} enthalten sind. Wir entfernen das Minimum id_{min} aus PQ_{min} und fügen es zusammen mit seiner Häufigkeit i in PQ_{min} als Priorität in eine zweite Priorityqueue PQ_{occ} ein, in der die Elemente absteigend nach ihren Häufigkeiten sortiert werden. In einem Vektor `pointers` der Länge k merken wir uns zudem für jede invertierte Liste die Position des aktuell zu betrachtenden Elements. Für die Liste(n), die das das aktuelle Minimum enthielt(en) erhöhen wir die Position um 1 und fügen das jeweils nächste Element in PQ_{min} ein.

```
1:    {2, 5, 7}
2:    {1, 2, 4, 5}
3:    {5}
4:    {2, 4, 5}
```

PQ_{occ}: {1 ↦ 1}

Erneut lassen wir uns aus PQ_{min} das Minimum ausgeben. Dieses ist die ID 2 und tritt insgesamt dreimal in PQ_{min} auf. Wir fügen also die ID 2 mit der Häufigkeit 3 in PQ_{occ} ein. Da es insgesamt häufiger vorkommt als die zuvor eingefügte ID 1 wird es in PQ_{occ} zuoberst eingereiht. Wir löschen alle drei Vorkommen von der ID 2 in PQ_{min} und fügen die jeweils folgenden Elemente aus den Listen ein.

```
1:    {2, 5, 7}
2:    {1, 2, 4, 5}
3:    {5}
4:    {2, 4, 5}
```

PQ_{occ}: {2 ↦ 3, 1 ↦ 1}

Nun ist die ID 4 das Minimum in PQ_{min}. Da es insgesamt zweimal vorkommt, wird es in PQ_{occ} zwischen der ID 2 und 1 eingereiht. Nachdem wir beide Vorkommen der ID 4 aus PQ_{min} entfernt haben, fügen wir die jeweils nächsten ID's aus den Listen ein:

```
1:    {2, 5, 7}
2:    {1, 2, 4, 5}
3:    {5}
4:    {2, 4, 5}
```

PQ_{occ}: {2 ↦ 3, 4 ↦ 2, 1 ↦ 1}

Diesmal kommt die minimale ID 5 in allen Listen vor, weshalb es beim Einfügen zur obersten ID in PQ_{occ} wird. Haben wir die ID's in PQ_{min} gelöscht, können wir nur noch die verbleibende ID 7 einfügen:

```
1:    {2, 5, 7}
2:    {1, 2, 4, 5}
3:    {5}
4:    {2, 4, 5}
```

PQ_{occ}: {5 ↦ 4, 2 ↦ 3, 4 ↦ 2, 1 ↦ 1}

Es ist logischerweise zugleich das Minimum, weshalb wir es mit der Häufigkeit 1 in PQ_{occ} einfügen. Da dort bereits ein Element mit der gleichen Häufigkeit vorhanden ist (nämlich die

ID 1) wird sie der Reihenfolge entsprechend nach dieser ID eingefügt. Da es keine Elemente mehr gibt, die wir in in PQ_{min} einfügen können, ist die Vereinigung der Listen abgeschlossen. Zugleich ist PQ_{occ} vollständig und wir können an ihr die häufigsten ID's ablesen.

$$PQ_{occ}\texttt{: \{5} \mapsto \texttt{4, 2} \mapsto \texttt{3, 4} \mapsto \texttt{2, 1} \mapsto \texttt{1, 7} \mapsto \texttt{1\}}$$

L_C ergibt sich aus der Reihenfolge der Elemente in PQ_{occ}. Für die Analyse der Laufzeit dieser Methode lassen sich folgende Beobachtungen machen:

Sei wie oben k die Anzahl der zu vereinigenden Listen und enthalte jede Liste jeweils m Elemente. PQ_{min} enthält stets höchstens k Elemente und jedes Element wird genau einmal in PQ_{min} eingefügt sowie genau einmal aus PQ_{min} gelöscht. Beide Operationen (Einfügen und Löschen) benötigen Zeit in $O(\log k)$. Es ergibt sich eine Laufzeit von $O(km * 2 * \log k) = O(km * \log k)$. Die Positionen innerhalb des `pointers`-Vektor müssen wir höchstens $(k * m)$-mal nachschlagen und inkrementieren. Da daraus eine lineare Laufzeit in der Anzahl der Elemente resultiert, ändert sich die Gesamtkomplexität nicht.

Zusätzlich müssen wir noch die Laufzeiten der Einfügeoperationen in PQ_{occ} berücksichtigen: Im ungünstigsten Fall (nämlich genau dann, wenn alle $k * m$ Elemente verschieden voneinander sind) muss jedes Element auch in PQ_{occ} eingefügt werden. Die Laufzeit für das Bestimmen des Elements mit der größten Häufigkeit in PQ_{occ} benötigt insgesamt also eine Laufzeit in $O(km \log km)$. Es ergibt sich für das Bestimmen des Elements, das am häufigsten in den k invertierten Listen vorkommt, eine Laufzeit von $O(km \log k) + O(km \log km) = O(km \log km)$.

Man könnte nun vermuten, dass das erste Element in L_C die ID des gesuchten Eintrags E_Q ist. In vielen Fällen wird dies auch der Fall sein, allerdings können wir uns aufgrund verschiedener Faktoren darauf nicht verlassen. Ein Faktor sind mögliche Fehler bei der Extraktion eines Titels bzw. von Referenzen. Enthält ein Wort aus Q z.B. aufgrund einer nicht lesbaren Zeichenkodierung einen Extraktionsfehler, so werden wir das Wort wahrscheinlich nicht im invertierten Index finden, weil es dort ohne Fehler gespeichert ist. Alle Einträge in DB_L, die dieses Wort in ihren Metadaten enthalten, werden deshalb bei einer Suche mit Q nicht berücksichtigt und im Kontext von Q unter Umständen in L_C zu niedrig eingeordnet. Der gleiche Fall tritt bei Wörtern mit alternativen Schreibweisen ein: Es werden nur diejenigen Einträge aus DB_L berücksichtigt, die das Wort mit der Schreibweise aus Q enthalten. Alle anderen Publikationen, die das Wort in einer alternativen Schreibweise enthalten, werden hingegen nicht berücksichtigt und werden möglicherweise ebenfalls in L_C zu niedrig eingeordnet. Ein weiterer Faktor ist die Existenz von mehreren Publikationen mit ähnlichen Titeln: Es ist durchaus möglich, dass es mehrere Titel wissenschaftlicher Publikation gibt, die entweder vollständig übereinstimmen oder sich z.B. lediglich in ihren Stoppwörtern unterscheiden. Da wir Stoppwörter nicht indiziert haben, würden die ID's der entsprechenden Einträge mit der gleichen Häufigkeit in L_C auftreten und gleichberechtigt behandelt werden, obwohl sie unterschiedliche Publikationen repräsentieren. Wir führen deshalb eine zusätzliche Bewertung für eine definierte Anzahl von *Kandidaten* $C \subseteq E$ (z.B. die 50 ersten Elemente aus L_C) durch, um den repräsentierenden Eintrag in DB_L mit einer fehlertoleranten Suche über Q identifizieren zu können.

Genau in dieser Bewertung unterscheidet sich die Titel-Zuordnung von der Referenzen-Zuordnung. Je nachdem, ob Q einen extrahierten Titel oder eine extrahierte Referenz enthält, haben wir es mit unterschiedlichen Typen von Informationen zu tun. Beide unterscheiden sich sowohl in der Menge enthaltener Informationen (eine Referenz beinhaltet neben dem Titel einer wissenschaftlichen Publikation zusätzlich auch Informationen zu Autoren, Erscheinungsjahr, Konferenz, etc.) als auch in der Eindeutigkeit der Daten (bei einer Referenz können wir z.B. nicht bestimmen, zu welchem Feld eine Teilzeichenkette von Q gehört). Um beiden Typen gerecht zu werden, unterscheiden wir sie in der Bewertungsfunktion der Kandidaten C.

4.3 Die Zuordnung von Titeln

Bei der Titel-Zuordnung möchten wir denjenigen Eintrag $C_i \in C$ bestimmen, dessen Titel am ähnlichsten zur Suchanfrage Q ist, die einen extrahierten Titel enthält. In den meisten Fällen ist zwar davon auszugehen, dass es einen Kandidaten $C_i \in C$ gibt, der exakt den extrahierten

Titel enthält und deshalb eine exakte Suche ausreichen würde. Bei möglichen Fehlern entweder bei der Titel-Identifizierung oder bei der Zeichen-Extraktion würde solch eine Suche aber fehlschlagen. Wir wählen deshalb eine fehlertolerante Suchmethode, mit der wir den zu Q ähnlichsten Kandidaten als E_Q identifizieren.

Die Bewertung der Kandidaten

Wir haben bereits in Kapitel 2 besprochen, dass die Ähnlichkeit zweier Zeichenketten X und Y über die sogenannte *Levenshtein-Distanz* in Zeit $O(|X||Y|)$ berechnet werden kann. Zur Beschleunigung dieses Verfahrens können zusätzliche Suchindexe (wie z.B. Suffixbäume) über den zu durchsuchenden Text gebaut werden. Die Erstellung solcher Indexe lohnen sich meist aber nur, wenn der zu durchsuchende Text sehr groß ist und zudem sehr oft durchsucht werden muss. Da sich in unserem Fall mit jeder Suchanfrage auch die zu durchsuchende Kandidatenmenge ändert und zudem die Längen der konstanten Anzahl von Kandidaten vergleichsweise kurz sind, sind beide Bedingungen bei uns nicht gegeben. Die Berechnung solch eines Index lohnt sich für unsere Anwendung deshalb nicht. Wir wählen stattdessen die „klassische" Methode und berechnen die Ähnlichkeit zwischen Q und dem Titel von $C_i \in C$, indem wir ihre Levenshtein-Distanz mit dem Algorithmus von *Needleman & Wunsch* berechnen.

Definition (Levensthein-Distanz)**.** *Unter der Levensthein-Distanz $d_L(X, Y)$ zwischen zwei Zeichenketten X und Y versteht man die minimale Anzahl von Editieroperationen, die notwendig ist, um A in B umzuwandeln. Mögliche Editieroperationen sind das Einfügen, das Löschen und das Ersetzen eines Zeichens in X oder Y.*

Bezeichne N_i das Präfix der Länge i von einer Zeichenkette N (mit $0 \leq i \leq |N|$) und $N[j]$ das Zeichen an Position i in N (mit $1 \leq j \leq |N|$). Die Levenshtein-Distanz $d_L(X, Y) = d_L(X_{|X|}, Y_{|Y|})$ zwischen den Zeichenketten X und Y können wir wie folgt berechnen:

$$
\begin{aligned}
d_L(X_0, Y_0) &= 0 \\
d_L(X_i, Y_0) &= i \\
d_L(X_0, Y_j) &= j \\
d_L(X_i, Y_j) &= \min \begin{cases}
d_L(X_i, Y_{j-1}) + 1 & \text{Einfügen des Zeichens } Y[j] \\
d_L(X_{i-1}, Y_j) + 1 & \text{Löschen des Zeichens } X[i] \\
d_L(X_{i-1}, Y_{j-1}) + 1 & \text{Ersetzen des Zeichens } X[i] \text{ durch } Y[j] \\
d_L(X_{i-1}, Y_{j-1}) + 0 & \text{wenn } X[i] = Y[j]
\end{cases}
\end{aligned}
$$

Beispiel 4.5 zeigt beispielhaft die Berechnung der Levenshtein-Distanz für die Zeichenketten X = `MATCHING` und Y = `LAUGHING`. Jede Zelle (i, j) enthält dabei genau den Wert $d_L(A_i, B_j)$. Die Levenshtein-Distanz zwischen X und Y steht demnach in der Zelle $(|X|, |Y|) = (8, 8)$ und beträgt 3. Das bedeutet, dass die Zeichenkette `MATCHING` mit 3 Editieroperationen in die Zeichenkette `LAUGHING` überführt werden kann (z.B. durch das Ersetzen von `M` durch L, von `T` durch `U` und von `C` durch `G`. Um zu einer Suchanfrage Q den gesuchten Eintrag E_Q zu finden, berechnen wir für jeden Kandidaten C_i aus der Kandidatenmenge C die Levensthein-Distanz $d_L(X, Y)$, mit $X = Q$ und $Y = t(C_i)$, wobei $t(C_i)$ den Titel von C_i bezeichne. Es gilt:

$$E_Q = \arg\min_{C_i \in C}(d_L(Q, t(C_i)))$$

wenn $d_L(Q, t(C_i))$ kleiner als die Hälfte der maximal erreichbaren Distanz ist, d.h. wenn Q und $t(C_i)$ in mehr als der Hälfte aller Zeichen übereinstimmen. Liegt die Distanz über diesem Schwellenwert, entscheiden wir, dass Q und $t(C_i)$ zu unterschiedlich zueinander sind. Sollte kein Kandidat den Schwellenwert unterschreiten, so gehen wir davon aus, dass kein repräsentierender Eintrag für Q in DB_L existiert und wir ordnen Q keinem Eintrag zu.

	ϵ	M	A	T	C	H	I	N	G
ϵ	0	1	2	3	4	5	6	7	8
L	1	1	2	3	4	5	6	7	8
A	2	2	1	2	3	4	5	6	7
U	3	3	2	2	3	4	5	6	7
G	4	4	3	3	3	4	5	6	6
H	5	5	4	4	4	3	4	5	6
I	6	6	5	5	5	4	3	4	5
N	7	7	6	6	6	5	4	3	4
G	8	8	7	7	7	6	5	4	3

Beispiel 4.5: Die Berechnung der Levenshtein-Distanz zwischen den Zeichenketten `MATCHING` und `LAUGHING` mit dem Algorithmus von *Needleman & Wunsch*. ϵ stehe für das leere Wort.

4.4 Die Zuordnung von Referenzen

Bei der Titel-Zuordnung konnten wir die Suchanfrage Q mit einem bestimmten Feld (nämlich mit dem Feld `title`) eines Kandidaten C_i vergleichen. Dies ist bei der Referenzen-Zuordnung problematisch, weil eine extrahierte Referenz nicht nur den Titel, sondern auch weitere Informationen wie z.B. die Autoren oder Angaben zur Konferenz enthält und wir die Werte der Metadaten-Felder in Q nicht ohne Weiteres identifizieren können. Da es keine allgemeingültigen Normen für den Aufbau von Referenzen gibt, müssen wir zusätzlich damit rechnen, dass Teile der Angaben in C_i im Vergleich zu den Angaben in der Literaturdatenbank fehlen, hinzugefügt wurden oder gar fehlerhaft sind. Um diesen Tatsachen gerecht zu werden, wählen wir eine zur Titel-Zuordnung alternative Bewertungsfunktion für das Bewerten der Kandidaten.

Die Bewertung der Kandidaten

Wir berechnen für jeden Kandidaten $C_i \in C$ insgesamt drei verschiedene Scores: einen Autoren-Score $s_A(C_i)$, einen Titel-Score $s_T(C_i)$ und einen Score $s_Y(C_i)$ für das Erscheinungsjahr des Kandidaten.

Mithilfe des Autoren-Scores können wir einerseits Publikationen mit dem gleichen Titel unterscheiden und andererseits alle Publikationen, die keinen Autoren mit Q gemeinsam haben, von allen weiteren Berechnungen ausschließen, da sie für E_Q nicht in Frage kommen. Zur Berechnung von $s_A(C_i)$ überprüfen wir mit einer exakten Suche, wie viele Nachnamen der Autoren von C_i in Q vorkommen. $s_A(C_i)$ berechnet sich dann aus dem prozentualen Anteil gefundener Autoren: von allen weiteren Berechnungen ausschließen, da sie für E_Q nicht in Frage kommen. Zur Berechnung von $s_A(C_i)$ überprüfen wir mit einer exakten Suche, wie viele Nachnamen der Autoren von C_i in Q vorkommen. $s_A(C_i)$ berechnet sich dann aus dem prozentualen Anteil gefundener Autoren:

$$s_A(C_i) = \frac{\text{\# gefundener Autoren von } C_i \text{ in } Q}{\text{\# Autoren von } C_i}$$

Mit dem Autoren-Score $s_A(C_i)$ filtern wir alle diejenigen Kandidaten heraus, die für eine weitere Betrachtung nicht in Frage kommen, weil sie keinen Autoren mit Q gemeinsam haben. Die vergleichsweise zeitaufwändige Berechnung des Titel-Scores führen wir also für einen Kandidaten C_i nur dann aus, wenn $s_A(C_i) > 0$.

Die Idee hinter dem Titel-Score $s_T(C_i)$ ist die gleiche wie bei der Titel-Zuordnung: Für jeden Kandidaten C_i berechnen wir die Ähnlichkeit zwischen seinem Titel und Q. Da Q aber nicht nur den Titel einer Publikation enthält, macht die Berechnung der Ähnlichkeit zwischen den beiden Zeichenketten über die Levenshtein-Distanz wenig Sinn, da eine zu große Distanz resultieren würde. Stattdessen suchen wir mit einem lokalen Ähnlichkeits-Score diejenige Region in Q, die mit dem Titel von C_i die größte Ähnlichkeit aufweist.

Definition (Lokaler Ähnlichkeits-Score)**.** *Unter einem lokalen Ähnlichkeits-Score $s(X, Y)$ versteht man eine Bewertung der Ähnlichkeit zwischen zwei Teilzeichenketten der Zeichenketten X und Y. Bezeichne N_i das Präfix der Länge i einer Zeichenkette N (mit $0 \leq i \leq |N|$) und $N[j]$ das Zeichen an*

Position i in N (mit $1 \leq j \leq |N|$). Mit dem Algorithmus von Smith & Waterman [49] berechnet sich der lokale Ähnlichkeits-Score $s(X,Y) = d_L(X_{|X|}, Y_{|Y|})$ zwischen X und Y wie folgt:

$$
\begin{aligned}
s(X_0, Y_0) &= 0 \\
s(X_i, Y_0) &= 0 \\
s(X_0, Y_j) &= 0 \\
s(X_i, Y_j) &= \max \begin{cases}
s(X_i, Y_{j-1}) - 1 & \textit{Einfügen des Zeichens } Y[j] \\
s(X_{i-1}, Y_j) - 1 & \textit{Löschen des Zeichens } X[i] \\
s(X_{i-1}, Y_{j-1}) - 1 & \textit{Ersetzen des Zeichens } X[i] \textit{ durch } Y[j] \\
s(X_{i-1}, Y_{j-1}) + 2 & \textit{wenn } X[i] = Y[j] \\
0
\end{cases}
\end{aligned}
$$

Der Titel-Score $s_T(C_i)$ ist dann

$$s_T(C_i) = \max_{\substack{0 \leq i \leq |X| \\ 0 \leq j \leq |Y|}} (s(X_i, Y_j))$$

mit $X = Q$ und $Y = t(C_i)$.

Beispiel 4.6 zeigt die Berechnungen des lokalen Ähnlichkeits-Scores für die Zeichenketten `MATCHING` und `LAUGHING`. Der maximale lokale Ähnlichkeits-Score zwischen den beiden Zeichenketten ist 8, welcher durch die längste gemeinsame Zeichenkette `HING` hergestellt wurde.

	ϵ	M	A	T	C	H	I	N	G
ϵ	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0
A	0	0	2	1	0	0	0	0	0
U	0	0	1	1	0	0	0	0	0
G	0	0	0	0	0	0	0	0	2
H	0	0	0	0	0	2	1	0	1
I	0	0	0	0	0	1	4	3	2
N	0	0	0	0	0	0	3	6	5
G	0	0	0	0	0	0	2	5	8

Beispiel 4.6: Die Berechnung des lokalen Ähnlichkeits-Score zwischen den Zeichenketten `MATCHING` und `LAUGHING`. ϵ stehe für das leere Wort. Der optimale lokale Ähnlichkeits-Score ist 8 und wird durch die gemeinsame Teilzeichenkette `HING` verursacht.

Im Gegensatz zur Titel-Zuordnung, bei der wir die *Distanz* zwischen zwei Zeichenketten berechnet haben, berechnen wir mit $s_T(C_i)$ die (lokale) *Ähnlichkeit* zwischen Q und dem Titel eines Kandidaten C_i. Das bedeutet, dass wir $s_T(C_i)$ nicht minimieren, sondern maximieren wollen. Derjenige Kandidat C_i, für den $s_T(C_i)$ maximal ist, besitzt einen Titel, der zum in Q enthaltenen Titel am ähnlichsten ist.

Normalisierung von $s_T(C_i)$

Um alle drei Scores später miteinander vergleichen und besser verrechnen zu können, möchten wir $s_T(C_i)$ normalisieren, so dass wir einen Wert im Intervall $[0,1]$ erhalten. Sei $s_{T_{min}}(C_i)$ der minimal und $s_{T_{max}}(C_i)$ der maximal erreichbare Titel-Score für einen Kandidaten.
Wenn $s_{T_{max}}(C_i) > s_{T_{min}}(C_i)$, gilt:

$$s_{T_{min}}(C_i) \leq s_T(C_i) \leq s_{T_{max}}(C_i) \qquad \Leftrightarrow \qquad 0 \leq \frac{s_T(C_i) - s_{T_{min}}(C_i)}{s_{T_{max}}(C_i) - s_{T_{min}}(C_i)} \leq 1$$

Beweis: Sei $a = s_{T_{min}}(C_i)$, $b = s_{T_{max}}(C_i)$ und $x = s_T(C_i)$.

$$\begin{aligned} & a \leq x \leq b \\ \Leftrightarrow \quad & 0 \leq x - a \leq b - a \\ \Leftrightarrow \quad & 0 \leq \frac{x-a}{b-a} \leq 1 \qquad (\text{da } (b-a) > 0) \end{aligned}$$

□

Wir berechnen den normalisierten Titel-Score $s_{T_n}(C_i)$ also mit

$$s_{T_n}(C_i) = \frac{s_T(C_i) - s_{T_{min}}(C_i)}{s_{T_{max}}(C_i) - s_{T_{min}}(C_i)}$$

mit $s_{T_{min}}(C_i) = 0$ und $s_{T_{max}}(C_i) = 2 * \min(|Q|, |t(C_i)|)$ um für den Titel-Score einen Wert im Intervall [0,1] zu erhalten.

Um mögliche Zweideutigkeiten zu vermeiden, berechnen wir mit $s_Y(C_i)$ einen weiteren Score, der die Existenz des Erscheinungsjahres $year(C_i)$ des Kandidaten C_i in Q identifiziert. Sollte es für Q mehrere Kandidaten geben, für die wir den gleichen Autoren-Score $s_A(C_i)$ und den gleichen Titel-Score $s_T(C_i)$ berechnet haben, können wir die Kandidaten möglicherweise mithilfe von $s_Y(C_i)$ unterscheiden. $s_Y(C_i)$ kann lediglich zwei Werte annehmen:

$$s_Y(C_i) = \begin{cases} 1, & \text{wenn } year(C_i) \in Q \\ 0, & \text{sonst.} \end{cases}$$

Den Gesamt-Score $s(C_i)$ eines Kandidaten C_i berechnet sich folgendermaßen aus den drei Scores:

$$s(C_i) = \frac{3}{4} * s_T(C_i) + \frac{1}{4} * \left(\frac{3}{4} * s_A(C_i) + \frac{1}{4} * s_Y(C_i) \right)$$

In ihm spiegeln sich grundsätzlich die Wichtigkeiten der Scores wider: Da der Titel für uns das wichtigste Identifikationsmerkmal einer wissenschaftlichen Publikation ist, wird der Titel-Score mit einem Faktor von $\frac{3}{4}$ am höchsten gewichtet. Die Kombination aus dem Autoren-Score $s_A(C_i)$ und dem Score $s_Y(C_i)$ für das Erscheinungsjahr wird dementsprechend insgesamt mit dem Faktor $\frac{1}{4}$ gewichtet. Des Weiteren sind die Autoren einer Publikation ein wichtigeres Identifikationsmerkmal als das Erscheinungsjahr, weshalb bei der Verrechnung beider Scores $s_A(C_i)$ mit dem Faktor $\frac{3}{4}$ wiederum höher gewichtet wird als $s_Y(C_i)$ mit dem Faktor $\frac{1}{4}$. Der gesuchte Eintrag E_Q ergibt sich aus $s(C_i)$ wie folgt:

$$E_Q = \arg\max_{C_i \in C}(s(C_i))$$

wenn $s(C_i)$ größer als die Hälfte des maximal erreichbaren Scores $s_{max}(C_i)$ ist. Ähnlich wie bei der Titel-Zuordnung entscheiden wir anhand dieses Schwellenwertes, ob ein Kandidat C_i für E_Q in Frage kommt. Überschreitet keiner der Kandidaten diesen Schwellenwert, so gehen wir davon aus, dass in DB_L kein repräsentierender Eintrag für Q existiert und wir ordnen Q abermals keinem Eintrag zu. Der maximal erreichbare Score $s_{max}(C_i)$ errechnet sich folgendermaßen: Durch die Normalisierung des Titel-Scores $s_T(C_i)$ können aller drei Scores im besten Fall den Wert 1 annehmen. Es gilt:

$$\begin{aligned} s_{max}(C_i) &= \frac{3}{4} * 1 + \frac{1}{4} * \left(\frac{3}{4} * 1 + \frac{1}{4} * 1 \right) \\ &= 1 \end{aligned}$$

wobei $|Q|$ die Länge der Suchanfrage Q und $|t(C_i)|$ die Länge des Titels von Kandidat C_i sei. Der Score $s(C_i)$ eines Kandidaten C_i muss also größer als $0,5$ sein, damit C_i als der repräsentierende Eintrag E_Q in Frage kommt.

Kapitel 5

Experimente

In diesem Kapitel stellen wir die Ergebnisse unserer Experimente vor, die wir durchgeführt haben, um die Qualität und die Effizienz der Titel- und Referenzen-Extraktion (vgl. Kapitel 3.2 und Kapitel 3.3) sowie der Titel- und Referenzen-Zuordnung (vgl. Kapitel 4.3 und Kapitel 4.4) evaluieren zu können. Bei der Durchführung jeder Evaluation interessierte uns insbesondere die Korrektheit der erhaltenen Ergebnisse. Um mögliche Zusammenhänge zu der Effizienz unserer Algorithmen zu erkennen, haben wir zusätzlich die Laufzeiten unserer Algorithmen gemessen.

5.1 Testdatensätze

Alle Evaluationen führen wir an zwei Testdatensätzen, bestehend aus PDF-Dateien wissenschaftlicher Publikationen, durch. Der erste Testdatensatz besteht aus insgesamt 700 PDF-Dateien zufällig ausgewählter Publikationen aus der Literaturdatenbank DBLP. Alle 700 Publikationen stammen aus den Jahren 2000 bis 2011, jeweils 60 aus den Jahren 2000 bis 2010 und 40 aus dem Jahr 2011. Der zweite Testdatensatz besteht aus insgesamt 500 PDF-Dateien ebenfalls zufällig ausgewählter Publikationen aus der Literaturdatenbank Medline. Er setzt sich aus jeweils 50 Publikationen aus den Jahren 2001 bis 2010 zusammen.

- **DBLP:** DBLP enthält Einträge zu mehr als 1,7 Millionen Publikationen aus dem Bereich der Informatik. Jeder Eintrag enthält neben den Angaben zu Titel, Autoren und Erscheinungsjahr einer Publikation unter anderem den Namen der Konferenz, auf der die Publikation veröffentlicht wurde. Jeder Eintrag ist eindeutig durch einen DBLP-Key identifizierbar. Alle Einträge sind in der rund 850 MB großen XML-Datei `dblp.xml` abgespeichert.
- **Medline:** Medline enthält Einträge zu mehr als 19 Millionen Publikationen aus allen Bereichen der Medizin. Jeder Eintrag enthält in seinen zahlreichen Metadaten unter anderem Informationen zu Titel, Autoren und Erscheinungsjahr der Publikation. Jeder Eintrag ist eindeutig über einen Medline-Key identifizierbar. Alle Einträge, die ursprünglich in insgesamt 650 XML-Dateien gespeichert waren, haben wir in eine XML-Datei namens `medline.xml` zusammengefasst, die etwa 5,3 GB groß ist und somit mehr als sechs mal so groß ist wie `dblp.xml`.

Beide Literaturdatenbanken haben wir bereits in Kapitel 4.1 ausführlich vorgestellt.

Für jede der insgesamt 1200 Publikationen haben wir den Titel manuell bestimmt (d.h. wir haben jede PDF-Datei geöffnet und den Titel der Publikation herauskopiert) und jeweils den DBLP-Key bzw. Medline-Key des repräsentierenden Eintrags in der Literaturdatenbank mithilfe der Suchmaschinen *CompleteSearch* [6] und *PubMed* [18] ermittelt. Für 1029 Publikationen (700 DBLP- und 329 Medline-Publikationen) haben wir das gleiche für die Referenzen getan, d.h. wir haben die Referenzen der Publikationen manuell bestimmt und für 1000 zufällig ausgewählter Referenzen (500 DBLP- und 500 Medline-Referenzen) den Key des entsprechenden Eintrags in der Literaturdatenbank ermittelt.

```
04411572.pdf   A Cross-layer Proportional Fair Sche[...]   conf/globecom/HuangN07
04587354.pdf   A Parallel Decomposition Solver for [...]   conf/cvpr/HazanMS08
04587380.pdf   A Learning-based Hybrid Tagging and [...]   conf/cvpr/YanNC08
04587382.pdf   Annotating Collections of Photos Usi[...]   conf/cvpr/CaoLKH08
04587464.pdf   A Scalable Graph-Cut Algorithm for N[...]   conf/cvpr/DelongB08
04587498.pdf   3D Pose Refinement from Reflections [...]   conf/cvpr/LaggerSLF08
04587505.pdf   An Adaptive Learning Method for Targ[...]   conf/cvpr/ChenLHC08
04587519.pdf   A Recursive Filter For Linear System[...]   conf/cvpr/TyagiD08
04587528.pdf   3D-2D Spatiotemporal Registration fo[...]   conf/cvpr/WangLL08
04587529.pdf   Accurate Eye Center Location Approac[...]   conf/cvpr/ValentiG08
[...]
```

Beispiel 5.1: Auszug aus der Ground-Truth-Datei `titles.dblp.qrels`. Jede Zeile enthält den Namen der zu bearbeitenden Publikation mit dem zu extrahierenden Titel und den DBLP-Key des Eintrags, der die Publikation repräsentiert.

Alle gesammelten Daten haben wir in sogenannten Ground-Truth-Dateien gespeichert, die für unsere Algorithmen zu gegebenen Eingaben die erwarteten Ausgaben definieren:

- `titles.dblp.qrels / titles.medline.qrels`**:** Mithilfe dieser Ground-Truth-Dateien evaluieren wir die Extraktion und die Zuordnung des Titels aus PDF-Dateien von DBLP- bzw. Medline-Publikationen. Für jede zu bearbeitende Publikation enthalten die Dateien eine Zeile, die nach dem Muster

  Name der PDF-Datei <TAB> *zu extrahierender Titel* <TAB> *Key*

  aufgebaut ist. Der Name der PDF-Datei identifiziert die zu bearbeitende Publikation. Getrennt durch ein Tabulator-Zeichen (<TAB>, dieses dient zur Unterscheidung der Elemente in einer Zeile) folgt der korrekte, manuell bestimmte Titel der Publikation. Wir erwarten von der Titel-Extraktion, dass sie genau diesen Titel extrahiert. Für eine spätere Evaluierung der Titel-Zuordnung enthält jede Zeile zudem den ermittelten Key des Eintrags in DBLP bzw. Medline. Genau diesen Key soll die Titel-Zuordnung zurückliefern. Falls kein entsprechender Eintrag existiert, verwenden wir statt dem Key das Schlüsselwort `NO_MATCH`. In solch einem Fall soll die Titel-Zuordnung keinen Key zurückliefern. Beispiel 5.1 zeigt einen Ausschnitt aus `titles.dblp.qrels`. Die Ground-Truth-Datei `titles.medline.qrels` ist äquivalent aufgebaut.

- `references.dblp.qrels / references.medline.qrels`**:** Diese Ground-Truth-Dateien verwenden wir, um die Referenzen-Extraktion und die Referenzen-Zuordnung zu evaluieren. Aufgrund der Menge der zu extrahierenden Daten unterscheiden sie sich in ihrem Aufbau von den Ground-Truth-Dateien der Titel-Extraktion und der Titel-Zuordnung:

  Name der PDF-Datei
  <TAB> *1. zu extrahierende Referenz* <TAB> *Key*
  <TAB> *2. zu extrahierende Referenz* <TAB> *Key*
  <TAB> *3. zu extrahierende Referenz* <TAB> *Key*
  ...

  Der Name der PDF-Datei identifiziert wieder die zu bearbeitende Publikation. In den folgenden Zeilen sind die in ihr enthaltenen Referenzen aufgelistet. Wir erwarten, dass die Referenzen-Extraktion genau diese Referenzen extrahiert. Um die Zeilen von jenen Zeilen zu unterscheiden, die den Namen einer PDF-Datei enthalten, ist ihnen jeweils ein <TAB>-Zeichen vorangestellt. Außerdem enthalten ausgewählte Referenzen den ermittelten Key desjenigen Eintrags, der die Referenz in der Literaturdatenbank repräsentiert. In Beispiel 5.2 ist ein Ausschnitt aus `references.medline.qrels` zu sehen, wobei `references.dblp.qrels` äquivalent aufgebaut ist.

Die Verwendung von zwei verschiedenen Testdatensätzen aus zwei verschiedenen Literaturdatenbanken gibt uns die Möglichkeit, die Titel- und Referenzen-Extraktion an möglichst

```
201-208.pdf
   American Psychiatric Association. Diagnostic and Statistical Mo[...]  NO_MATCH
   ANTHONY, J.C.; WARNER, L.A.; ANDKESSLER, R.C.Comparative epidem[...]  NO_MATCH
   KESSLER, R.C.; MCGONAGLE, K.A.; ZHAO, S.; ETAL. Lifetime and 1.[...]  8279933
   WARNER, L.A.; KESSLER, R.C.; HUGHES, M.; ANTHONY, J.C.;ANDNELSO[...]  7872850
   [...]
JVS001-02-03.pdf
   [1] Bredt, D.S and Snyder, S.H. Nitric oxide, a novel neuronal [...]  1370373
   [2] Cooper, J. and Walker, R.D. Listeriosis. Vet. Clin. North. [...]  9532671
   [3] Charlton, K.M. and Garcia, M.M. Spontaneous listeric encept[...]  560741
   [...]
12249_2008_Article_1255.pdf
   1. Yamashita F, Bando H, Koyama Y, Kitagawa S,Takakura Y, Hashi[...]  8165175
   2. Phillips CA, Michiniak BB. Transdermal delivery of drugs wi [...]  8748324
   3. Flynn GL, Yalkowsky SH, Roseman TJ. Masstransport phenomena [...]  4828694
   4. Ozawa T, Takahashi M. Skin hydration: Recent Advances. Acta [...]  8091923
   5. Iazzo PA, Olsen RA, Seewald MJ, Powis G, Stier A,Van Dyke RA[...]  2265428
   [...]
[...]
```

Beispiel 5.2: Auszug aus der Ground-Truth-Datei `references.medline.qrels` mit PDF-Dateinamen und Auflistung der zu extrahierenden Referenzen. Für die Referenzen-Zuordnung enthalten ausgewählte Referenzen den Medline-Key des Literaturdatenbank-Eintrages, der die Referenz repräsentiert.

vielen, unterschiedlich strukturierten Publikationen sowie die Titel- und Referenzen-Zuordnung an verschieden großen Literaturdatenbanken zu testen. Wir erhoffen uns dadurch besonders aufschlussreiche Informationen über die Korrektheit der Ergebnisse und über die Effizienz unserer Algorithmen.

5.2 Die Testumgebung

Während die Algorithmen der Titel- und Referenzen-Extraktion in der Programmiersprache *Java* unter OpenJDK 6 geschrieben wurden, wurden die Algorithmen für die Titel- und Referenzen-Zuordnung in *C++* implementiert und mit der *GNU Compiler Collection* GCC 4.4.1 mit dem *-O3 Flag* kompiliert.

Alle Experimente wurden auf einem Rechner mit vier Intel Xeon X5560 Prozessoren, jeweils mit 2,8 GHz Taktfrequenz und 8MB Cache-Speicher, sowie mit einem Hauptspeicher von 36.274 MB durchgeführt. Das zum Einsatz gekommene Betriebssystem war Ubuntu 9.10 Karmic Koala, 64Bit.

5.3 Evaluation der Extraktion von Titeln

Zur Evaluation der Titel-Extraktion lesen wir die Ground-Truth-Datei `titles.dblp.qrels` (bzw. `titles.medline.qrels`) zeilenweise ein und entnehmen jeder Zeile jeweils den Dateinamen der PDF-Datei und den zu extrahierenden Titel, den wir zunächst aber zurückhalten. Danach lassen wir den Titel aus der durch den Dateinamen definierten PDF-Datei durch unseren Algorithmus extrahieren und vergleichen ihn mit dem korrekten Titel. Der extrahierte Titel gilt dann als korrekt, wenn er sich höchstens in einzelnen Zeichen vom korrekten Titel unterscheidet. Wenn er dagegen mindestens ein Wort zu viel oder zu wenig enthält, so gilt die Titel-Extraktion als fehlgeschlagen.

Zusätzlich messen wir jeweils die Laufzeit, die unser Algorithmus für eine Titel-Extraktion benötigt. Aus der Summe aller Laufzeiten berechnen wir die durchschnittliche Laufzeit, indem wir aus ihr das arithmetische Mittel bestimmen. Um eine möglichst aussagekräftige Laufzeitanalyse zu erhalten, wiederholen wir die gesamte Evaluation der Titel-Extraktion zehnmal und bilden aus den erhaltenen durchschnittlichen Laufzeiten erneut das arithmetische Mittel.

Ergebnisse & Diskussion

Korrektheitsanalyse *Titel-Extraktion*	**DBLP**		**Medline**	
Anzahl zu bearbeitender PDF-Dateien	700		500	
Anzahl extrahierbarer Titel	692	*98,8%*	494	*98,8%*
Anzahl korrekt extrahierter Titel	663	*94,7%*	449	*89,8%*
Anzahl partiell extrahierter Titel	3	*0,4%*	17	*3,4%*
Anzahl überladen extrahierter Titel	0	*0,0%*	1	*0,2%*
Anzahl nicht extrahierter Titel	26	*3,7%*	27	*5,4%*

Tabelle 5.3: Ergebnisse der Korrektheitsanalyse für die Titel-Extraktion aus Publikationen von DBLP und Medline.

Wir haben die Titel-Extraktion an 700 Publikationen aus DBLP und an 500 Publikationen aus Medline getestet. Für 8 DBLP- und 6 Medline-Publikationen war die Titel-Extraktion generell nicht möglich, da sie von PDFBox zum Beispiel aufgrund einer unbekannten Zeichenkodierung nicht gelesen werden konnten. Die Anzahl maximal extrahierbarer Titel beläuft sich für DBLP also auf 692 und für Medline auf 494 (vgl. Tabelle 5.3).

Aus diesen konnten wir für 663 (das entspricht *94,7%* aller Publikationen aus dem DBLP-Testdatensatz) DBLP-Publikationen und für 449 (*89,8%* aller Publikation des Medline-Testdatensatzes) Medline-Publikationen den korrekten Titel extrahieren – für 29 DBLP- und 45 Medline-Publikationen schlug die Titel-Extraktion dagegen fehl. Fehlgeschlagene Titel-Extraktionen unterscheiden wir nochmals in die Fälle, in denen der Titel nur zum Teil extrahiert wurde („partiell extrahierte Titel“), in denen zuviele Informationen extrahiert wurden, der Titel aber vollständig enthalten war („überladen extrahierte Titel“) und in denen der Titel gar nicht enthalten war („nicht extrahierte Titel“).

Gründe für partiell extrahierte Titel fanden wir vor allem in der Struktur der Titel. Bei 1 DBLP- und 10 Medline-Publikationen fand z.B. innerhalb des Titels ein Wechsel der Schriftauszeichnung statt; so enthielten Überschriften aus Medline-Publikationen wissenschaftliche Symbole oder Begriffe, die durch eine alternative Schriftauszeichnung (z.B. durch Kursiv-Schriftarten oder verschiedene Schriftgrößen) im Gegensatz zu den restlichen Titel-Textzeilen hervorgehoben waren (vgl. Beispiel B.3 und Beispiel B.4). Dies war für die Extraktion der Titel deshalb problematisch, weil wir die Textzeilen u.a. nach ihren Schriftauszeichnungen gruppiert haben. Veränderungen in der Schriftauszeichnung hatten dabei die Einteilung der Textzeile in eine neue Gruppe zur Folge. Ein eigentlich zusammenhängender Titel wurde somit in verschiedene Gruppierungen aufgeteilt und nicht mehr als ein Element betrachtet. Da wir letztendlich immer nur eine Gruppierung als Titel bestimmen, wurde der Titel nur partiell extrahiert.

Zusätzlich waren bei insgesamt 5 DBLP- und 10 Medline-Publikationen die von PDFBox erhaltenen Attribute der `TextPosition`-Objekte so fehlerhaft, dass der Titel entweder partiell, überladen oder gar nicht extrahiert wurde. Dies war besonders bei sogenannten *OCR-Dateien* der Fall. Das Problem bei OCR-Dateien ist, dass der enthaltene Text zumeist eingescannt und mithilfe einer Texterkennungssoftware eingelesen wurde. Je nach Qualität des Scans und der Texterkennungssoftware können die Inhalte dieser Dateien entweder überhaupt nicht oder nur stark fehlerbehaftet gelesen werden. Für uns macht sich dies zum Beispiel durch fehlende Textzeilen, falsch erkannte Zeichen oder mangelhafte Positionsangaben bemerkbar. Sind wir dadurch nicht mehr in der Lage, alle Textzeilen des Titels von den restlichen Textzeilen zu unterscheiden, schlägt auch seine Extraktion fehl.

Die größten Probleme bereiteten uns jedoch Publikationen mit Textzeilen, welche eine größere Schriftgröße als die Titel-Textzeilen aufwiesen und zudem länger als der definierte Schwellenwert (20 Zeichen) waren (vgl. Beispiel B.1). In insgesamt 19 DBLP- und 22 Medline-Publikationen war dies der Fall, weshalb wir ihren Titel nicht extrahieren konnten. Durch die Existenz dieses Schwellenwertes, die eine Zeichenkette erreichen muss, um als Titel identifiziert werden zu können, entgingen uns zudem die Titel aus insgesamt 4 DBLP- und 3 Medline-

Publikationen, da sie allesamt kürzer als 20 Zeichen waren (vgl. Beispiel B.2).

Laufzeitanalyse *Titel-Extraktion*	**DBLP**		**Medline**	
Gesamtlaufzeit, pro PDF-Datei	43,46 ms		42,65 ms	
PDF-Datei laden	15,08 ms	*34,7%*	14,33 ms	*33,6%*
Zeichen extrahieren	22,25 ms	*51,2%*	22,32 ms	*52,3%*
Textzeilen zusammensetzen	6,05 ms	*13,9%*	5,93 ms	*13,9%*
Titel bestimmen	0,08 ms	*0,2%*	0,07 ms	*0,2%*

Tabelle 5.4: Detaillierte Ergebnisse der Laufzeitanalyse für die Titel-Extraktion aus Publikationen von DBLP und Medline.

Die Analyse der Laufzeiten für die Titel-Extraktion ergibt, dass die Laufzeiten für beide Testdatensätze sehr ähnlich sind: Die Titel-Extraktion dauert für DBLP-Publikationen im Durchschnitt insgesamt 43,46ms und für die Medline-Publikationen 42,65ms pro PDF-Datei (vgl. Tabelle 5.4). Eine Aufgliederung der Laufzeiten in die Teilaufgaben der Titel-Extraktion zeigt, dass lediglich das Laden der PDF-Dateien von Medline-Publikationen im Schnitt etwas schneller bewerkstelligt werden konnte (14,33ms pro PDF-Datei) als von DBLP-Publikationen (15,08ms pro PDF-Datei). Das ist erst nach einem Blick auf die durchschnittliche Dateigrößen der Publikationen aus beiden Datensätzen plausibel: Eine Medline-Publikation ist mit einer durchschnittlichen Dateigröße von 577 KB kleiner als eine durchschnittlich 619 KB große DBLP-Publikation und kann deshalb schneller geladen werden. Alle anderen Teilaufgaben (Zeichen extrahieren, aus den Zeichen Textzeilen zusammensetzen, aus den Textzeilen den Titel bestimmen) benötigen im Schnitt für beide Testdatensätze jeweils sehr ähnliche Laufzeiten.

Ein großer Nachteil im Hinblick auf die Laufzeit ist, dass wir jeweils die gesamte PDF-Datei laden müssen, obwohl wir lediglich den Inhalt der ersten Seite der PDF-Datei extrahieren wollen. Zwar können wir die anschließende Extraktion der Zeichen auf die erste Seite beschränken – trotzdem macht sie mit über 50% den Hauptanteil der Laufzeiten aus. Beide Aufgaben sind Aufgaben von PDFBox und nehmen mit über 85% den überwiegenden Anteil der Gesamtlaufzeit in Anspruch. Der Gedanke, diese Laufzeiten für unsere Bedürfnisse optimieren zu wollen, liegt deshalb nahe. Im Hinblick auf die äußerst umfangreichen und tiefgründigen PDF-Spezifikationen würde dieses Vorhaben den Rahmen dieser Masterarbeit aber deutlich überschreiten. Dennoch erwägen wir es für eine mögliche Weiterentwicklung der Algorithmen zu einem späteren Zeitpunkt.

Die in unserer Verantwortung liegenden Aufgaben, nämlich das Zusammensetzen der Textzeilen aus den extrahierten Zeichen und das Bestimmen des Titels aus diesen Textzeilen, verbrauchen vergleichsweise wenig Laufzeit. Mit ungefähr 6ms pro PDF-Datei macht das Zusammensetzen der Textzeilen im Schnitt lediglich rund 14% und das Identifizieren der Titel mit etwa 0,08ms rund 0,1% der durchschnittlichen Laufzeit aus.

5.4 Evaluation der Zuordnung von Titeln

Für die Evaluation der Titel-Zuordnung entnehmen wir den Zeilen aus `titles.dblp.qrels` und `titles.medline.qrels` den Titel und den zu bestimmenden Key key_{gt} (gt stehe für „ground truth“). Durch die Zuordnung des Titels durch unseren Algorithmus erhalten wir einen Key key_m (m stehe für „matched“), den wir mit key_{gt} vergleichen. Wir nennen das Ergebnis unseres Algorithmus genau dann als korrekt, wenn entweder key_{gt} und key_m übereinstimmen oder wenn key_{gt} das Schlüsselwort `NO_MATCH` ist und key_m keinen Wert enthält. Für die durchschnittliche Laufzeit haben wir die gesamte Evaluation der Titel-Zuordnung zehnmal wiederholt und aus den durchschnittlichen Laufzeiten das arithmetische Mittel gebildet.

Ergebnisse & Diskussion

Wir haben festgestellt, dass eine Titel-Zuordnung hauptsächlich dann fehlschlägt, wenn sich der repräsentierende Eintrag nicht unter den k zu bewertenden Kandidaten befindet. Die Ergebnisse der Korrektheitsanalyse hängen deshalb stark von der Anzahl k zu bewertender Kandidaten ab. Zur Erinnerung: Sowohl bei der Titel- als auch bei der Referenzen-Zuordnung bewerten wir nach der Vereinigung der invertierten Listen aus dem Index und der Sortierung der Elemente nach ihren Häufigkeiten, die k ersten Elemente der vereinigten Liste in einem zusätzlichen Bewertungsschritt (vgl. Kapitel 4.3 und Kapitel 4.4). Je größer k ist, desto wahrscheinlicher wird es zwar, dass sich der repräsentierende Eintrag unter den zu bewertenden Kandidaten befindet, aber desto länger dauert auch dessen Berechnung. Die Anzahl der Kandidaten beeinflusst deshalb auch die Ergebnisse der Laufzeitanalyse entscheidend. Je nachdem, ob man den Fokus auf eine möglichst hohe Zuordnungsrate oder eine möglichst schnelle Berechnung legen möchte, ist also ein entsprechend höherer oder niedrigerer Wert für k zu wählen. Um das Verhalten unserer Algorithmen für verschiedene Werte für k analysieren zu können, führen wir jedes Experiment einmal für $k = 50$ (Ergebnisse in Tabelle 5.5 und Tabelle 5.6) und einmal für $k = 500$ (Ergebnisse in Tabelle 5.7 und Tabelle 5.8) aus.

Korrektheitsanalyse *Titel-Zuordnung, k = 50*	**DBLP**		**Medline**	
Anzahl zuzuordnender Titel	700		500	
Korrekte Titel-Zuordnungen	687	*98,1%*	421	*84,2%*

Tabelle 5.5: Ergebnisse der Korrektheitsanalyse für die Titel-Zuordnung aus Publikationen von DBLP und Medline mit $k = 50$.

Für $k = 50$ erreichen wir bei den DBLP-Publikationen ein Ergebnis von 98,1% und bei den Medline-Publikationen ein Ergebnis von 84,2% korrekten Titel-Zuordnungen. Der deutlich niedrigere Wert für die Medline-Publikationen lässt sich durch die sehr viel größere Anzahl der in Medline enthaltenen Einträge erklären. Dadurch ist es wahrscheinlicher, dass es viele Publikationen mit einem ähnlichen Titel gibt, so dass sich der repräsentierende Eintrag nicht unter den ersten 50 Kandidaten befindet und somit nicht als der gesuchte Eintrag ausgewählt werden kann.

Laufzeitanalyse *Titel-Zuordnung, k = 50*	**DBLP**		**Medline**	
Gesamtlaufzeit, pro Titel	4,93 ms		26,31 ms	
Kandidaten finden	2,92 ms	*59,2%*	22,45 ms	*85,3%*
50 Kandidaten bewerten	2,01 ms	*40,8%*	3,86 ms	*14,7%*
1 Kandidaten auswählen	0,002 ms	*< 0,1%*	0,002 ms	*< 0,1%*

Tabelle 5.6: Detaillierte Ergebnisse der Laufzeitanalyse für die Titel-Zuordnung aus Publikationen von DBLP und Medline mit $k = 50$.

Die größere zu bearbeitende Datenmenge bei Medline wirkt sich zudem erheblich auf die Laufzeit der Titel-Zuordnung aus. So dauert die Titel-Zuordnung von Medline-Publikationen rund fünfmal länger (im Schnitt 26,31ms pro Titel) als bei DBLP-Publikationen (im Schnitt 4,93ms pro Titel, vgl. Tabelle 5.6). Besonders stark fällt dabei die Laufzeit zum Finden der Kandidaten ins Gewicht, also das Anfragen der invertierten Listen aus dem Index, ihrer Vereinigung sowie ihrer Sortierung nach der Häufigkeit ihrer enthaltenen Elemente. Der Grund dafür ist, dass Medline deutlich mehr Einträge enthält als DBLP (etwa zehnmal soviel) und dadurch die zu vereinigenden invertierten Listen durchschnittlich länger sind. Es müssen generell mehr ID's verarbeitet werden, was sich auf die Laufzeit auswirkt. Im Gegensatz dazu sind die Laufzeiten für die Bewertung der Kandidaten mit 2,01ms für DBLP-Publikationen bzw. 3,86ms für Medline-Publikationen vergleichbar, da hier jeweils mit $k = 50$ eine konstante Anzahl von Kan-

didaten bewertet wird. Die Laufzeit für die Bewertung der Kandidaten hängt also genauso wie die Laufzeit für das Bestimmen des repräsentierenden Eintrags aus den Kandidaten nicht von der Anzahl der Einträge in der Literaturdatenbank ab.

Korrektheitsanalyse *Titel-Zuordnung, k = 500*	**DBLP**	**Medline**
Anzahl zuzuordnender Titel	700	500
Korrekte Titel-Zuordnungen	699 *99,9%*	461 *92,2%*

Tabelle 5.7: Ergebnisse der Korrektheitsanalyse für die Titel-Zuordnung aus Publikationen von DBLP und Medline mit $k = 500$.

Für $k = 500$ konnten wir für die DBLP-Publikationen 99,9% der Titel korrekt zuordnen (vgl. Tabelle 5.7). Lediglich für eine Publikation schlug die Zuordnung des Titels fehl. Dies lag nicht daran, dass sich der entsprechende DBLP-Eintrag nicht unter den ersten 500 Kandidaten befand, sondern weil er einen falschen Titel der Publikation beinhaltete.

Während wir im Gegensatz zu den Experimenten für $k = 50$ bei den DBLP-Publikationen das ohnehin schon gute Ergebnis von 98,1% also nur minimal verbessern konnten, gelang uns für die Medline-Publikationen eine Verbesserung des Ergebnisses von 84,2% auf 92,2% korrekter Titel-Zuordnungen.

Laufzeitanalyse *Titel-Zuordnung, k = 500*	**DBLP**		**Medline**	
Gesamtlaufzeit, pro Titel	21,45 ms		56,46 ms	
Kandidaten finden	2,96 ms	*13,8%*	23,01 ms	*40,8%*
500 Kandidaten bewerten	18,48 ms	*86,2%*	33,44 ms	*59,2%*
1 Kandidaten auswählen	0,01 ms	*< 0,1%*	0,01 ms	*< 0,1%*

Tabelle 5.8: Detaillierte Ergebnisse der Laufzeitanalyse für die Titel-Zuordnung aus Publikationen von DBLP und Medline mit $k = 500$.

Leider geht diese Verbesserung der Korrektheit der Titel-Zuordnungen mit einem deutlichen Anstieg der Laufzeiten einher. Während das Finden der Kandidaten im Vergleich zu $k = 50$ für beide Testdatensätze nahezu die gleiche Zeit benötigt, hängt besonders die Laufzeit für das Bewerten der Kandidaten stark von k ab. Sie erhöht sich bei beiden Literaturdatenbanken ebenso wie k beinahe um den Faktor 10 (18,48ms für DBLP-Publikationen und 33,44ms für Medline-Publikationen). Dies hat zur Folge, dass die Gesamtlaufzeit der Titel-Zuordnung für DBLP-Publikationen im Schnitt auf 21,45ms und für Medline-Publikationen auf 56,46ms pro Titel ansteigt. Angesichts dieser deutlichen Verschlechterung der Laufzeiten ist eine sorgfältige Abwägung, ob der Schwerpunkt eher auf möglichst korrekte oder eher auf möglichst schnell berechnete Ergebnisse gelegt werden soll, sinnvoll. Dazu entscheiden wir, ob der bei einer Erhöhung von k erreichte Anstieg der Korrektheit der Ergebnisse den damit verbundenen Anstieg der Laufzeiten rechtfertigt.

Für den DBLP-Testdatensatz ist dies sicherlich nicht der Fall, da mit der Erhöhung von k auf 500 die Korrektheit der Ergebnisse lediglich um etwa 1,7% steigt (von 687 auf 699 Publikationen), die Laufzeit aber um etwa 335,1% steigt (von 4,93ms auf 21,45ms). Für den Medline-Testdatensatz steigt die Korrektheit der Ergebnisse dagegen um etwa 9,5% (von 421 auf 461 Publikationen), während die Laufzeit nur um etwa 114,6% steigt (von 26,31ms auf 56,46ms). Im Vergleich zum DBLP-Datensatz lohnt sich die Erhöhung von k für den Medline-Datensatz also mehr. Ob sich die Erhöhung von k prinzipiell lohnt, hängt auch von den Ergebnissen der Evaluation zur Referenzen-Zuordnung ab.

5.5 Evaluation der Extraktion von Referenzen

Für die Evaluation der Referenzen-Extraktion lesen wir aus `references.dblp.qrels` (bzw. `references.medline.qrels`) zu jedem Dateinamen $filename_{gt}$ die zu extrahierenden Referenzen $references_{gt}$ aus und lassen unseren Algorithmus die Referenzen aus der entsprechenden PDF-Datei extrahieren. Wir erhalten eine Liste $references_{ex}$ (*ex* steht für „extracted"), die die extrahierten Referenzen als Zeichenketten enthält. Wenn eine Referenz aus $references_{ex}$ auch in $references_{gt}$ enthalten ist, wurde sie korrekt extrahiert. Für alle extrahierten Zeichenketten, die nicht in $references_{gt}$ enthalten sind, schlug die Extraktion dagegen fehl.

Ergebnisse & Diskussion

Korrektheitsanalyse *Referenzen-Extraktion*	**DBLP**		**Medline**	
Anzahl zu bearbeitender PDF-Dateien	700		329	
Anzahl enthaltener Referenzen	9.753		10.215	
Anzahl extrahierbarer Referenzen	9.609	*98,5%*	10.135	*99,2%*
Anzahl korrekt extrahierter Referenzen	7.958	*81,6%*	9306	*91,1%*
Anzahl Referenzen aus OCR-Dateien	1.469	*15,1%*	218	*2,1%*

Tabelle 5.9: Ergebnisse der Korrektheitsanalyse für die Referenzen-Extraktion aus Publikationen von DBLP und Medline.

Die Referenzen-Extraktion haben wir an insgesamt 700 DBLP- und 329 Medline-Publikationen getestet. Eine interessante Beobachtung war, dass die Medline-Publikationen insgesamt mehr Referenzen enthielten (nämlich 10.215 Referenzen), als die DBLP-Publikationen (9.753 Referenzen), obwohl wir mehr als doppelt so viele DBLP- wie Medline-Publikationen verwendet haben. Im Durchschnitt enthält eine Medline-Publikation also deutlich mehr Referenzen ($\sim$ 31 Referenzen pro Publikation), als eine DBLP-Publikation ($\sim$ 14 Referenzen pro Publikation). Insgesamt konnten wir aus den DBLP-Publikationen 81,6% und aus den Medline-Publikationen 91,1% der Referenzen korrekt extrahieren (vgl. Tabelle 5.9). Die 8 DBLP- bzw. 6 Medline-Publikationen, die bereits bei der Titel-Extraktion nicht gelesen werden konnten, enthielten 162 bzw. 80 Referenzen, die wir nicht extrahieren konnten. Die Anzahl extrahierbarer Referenzen belief sich also für den DBLP-Testdatensatz auf 9.609 und für den Medline-Testdatensatz auf 10.135 Referenzen.

Ähnlich wie bei der Titel-Extraktion beeinträchtigten OCR-Dateien vor allem bei den DBLP-Publikationen das Ergebnis erheblich. Im DBLP-Testdatensatz existierten 132 solcher OCR-Dateien, die insgesamt 1.469 Referenzen enthielten und im Medline-Testdatensatz 14 OCR-Dateien, die 218 Referenzen enthielten.

Die Auswirkungen solcher OCR-Dateien sind auf die Ergebnisse der Referenzen-Extraktion sehr viel größer als bei der Titel-Extraktion, weil wir für die Referenzen-Extraktion generell mehr Daten extrahieren und wir uns deshalb mit mehr fehlerhaften Daten konfrontiert sehen. Insbesondere Referenz-Anker wurden in OCR-Dateien häufig entweder nicht als solche erkannt oder als einzelnes Element extrahiert, also nicht in den Kontext zur zugehörigen Textzeile gesetzt. Beispiele für erhaltene Extraktionen des Referenz-Ankers `"[1]"` sind beispielsweise[1]:

```
[lI, [I, [ 1l, 1I1, [ 11, [1J, (1I
```

Dadurch war es möglich, dass die Referenz-Anker nur für manche Referenz-Titel erkannt wurden, was eine zuverlässige Erkennung der Referenz-Titel unmöglich machte (z.B. wurden Referenz-Titel als Referenz-Anhänge bzw. Referenz-Anhänge als Referenz-Titel erkannt). Auch

[1] Das Zeichen `l` stellt den Kleinbuchstaben von `L` dar, das Zeichen `I` den Großbuchstaben von `i` und das Zeichen `1` die Zahl eins.

war die Unterscheidung der Referenzen anhand ihrer Zeilenabstände untereinander im Allgemeinen so gut wie nicht anwendbar, weil es durch die Ungenauigkeit der Positionsangaben und den damit zu berücksichtigenden Toleranzen kaum möglich war, zu entscheiden, ob eine gerade betrachtete Textzeile noch zur aktuell betrachteten Referenz gehört, eine neue Referenz einleitet, oder das Ende des Literaturverzeichnisses darstellt. So schlug die Referenzen-Extraktion meistens auch für diejenigen Literaturverzeichnisse fehl, bei denen weder Referenz-Titel Anker enthielten noch Referenz-Anhänge eingerückt waren. Das Fehlen einer Überschrift des Literaturverzeichnisses führte außerdem dazu, dass wir das Literaturverzeichnis nicht erkannt haben und aufgrund der Vorgehensweise unserer Heuristik keine der Referenzen identifizieren konnten.

Schwierigkeiten hatte unser Algorithmus zudem mit Literaturverzeichnissen, die durch verschiedenartige Einschübe (z.B. Tabellen oder Abbildungen) unterbrochen waren (vgl. Beispiel B.5) und nach den Einschüben weitere Referenzen folgten. In solchen Fällen wurde entweder fälschlicherweise das Ende des Literaturverzeichnisses diagnostiziert und somit zu wenig Referenzen extrahiert oder die Einschübe als Teil des Literaturverzeichnisses identifiziert und somit zu viele Informationen extrahiert. Das Konzept der Content-Box zum Ausschluss von Kopf- und Fußzeilen, die bei der Referenzen-Extraktion vor allem bei Seitenwechseln von Bedeutung sind, funktionierte hingegen in der Regel problemlos, so dass Kopf- und Fußzeilen nicht extrahiert wurden.

Laufzeitanalyse *Referenzen-Extraktion*	**DBLP**		**Medline**	
Gesamtlaufzeit, pro PDF-Datei	141,85 ms		170,75 ms	
PDF-Datei laden	14,84 ms	*10,5%*	13,43 ms	*7,9%*
Zeichen extrahieren	92,25 ms	*65,0%*	109,6 ms	*64,2%*
Textzeilen zusammensetzen	29,17 ms	*20,6%*	41,14 ms	*24,1%*
Referenzen bestimmen	5,59 ms	*3,9%*	6,58 ms	*3,8%*

Tabelle 5.10: Detaillierte Ergebnisse der Laufzeitanalyse für die Referenzen-Extraktion aus Publikationen von DBLP und Medline.

Zur Extraktion der Referenzen benötigte unser Algorithmus für die DBLP-Publikationen im Schnitt 141,85ms und für die Medline-Publikationen 170,75ms pro PDF-Datei (vgl. Tabelle 5.10). Erwartungsgemäß sind die Laufzeiten für das Laden der PDF-Dateien (im Schnitt 14,84ms für eine PDF-Datei einer DBLP-Publikation und 13,43ms für eine PDF-Datei einer Medline-Publikation) mit den entsprechenden Laufzeiten aus der Titel-Extraktion vergleichbar, da bei beiden Vorgängen die kompletten Dateien geladen werden müssen. Anders als bei der Titel-Extraktion, bei der wir die Textzeilen nur von der ersten Seite der Publikation extrahieren, untersuchen wir bei der Referenzen-Extraktion jeweils die komplette PDF-Datei und extrahieren alle Zeichen, die wir zu Textzeilen zusammensetzen. Dies wirkt sich deutlich auf die Laufzeiten dieser Aufgaben aus: Sowohl die Extraktion der Zeichen als auch das Zusammensetzen der Textzeilen nimmt mehr Zeit in Anspruch als bei der Titel-Extraktion. Auffällig ist dabei, dass beide Aufgaben im Schnitt für DBLP-Publikationen schneller erledigt werden (92,25ms und 29,17ms pro PDF-Datei), als für Medline-Publikationen (109,6ms und 41,14ms pro PDF-Datei). Das ist ein Indiz dafür, dass Medline-Publikationen nicht nur mehr Referenzen, sondern auch generell mehr Text beinhalten. Somit müssen bei DBLP-Publikationen durchschnittlich weniger Referenzen aus den Textzeilen bestimmt werden, weshalb diese Aufgabe mit 5,59ms pro Publikation rund eine Millisekunde kürzer als bei Medline-Publikationen (6,58ms pro Publikation) dauert.

5.6 Evaluation der Zuordnung von Referenzen

Zur Evaluation der Referenzen-Zuordnung lesen wir die beiden Ground-Truth-Dateien `references.dblp.qrels` und `references.medline.qrels` zeilenweise ein und lassen jede Referenz $reference_{gt}$ durch unseren Algorithmus einem Eintrag in DBLP bzw. Medline zuordnen.

Falls ein geeigneter Eintrag gefunden wurde, erhalten wir einen Key key_m, den wir mit key_{gt} aus `references.matching.qrels` vergleichen. Sind beide äquivalent, hat unser Algorithmus den korrekten Literaturdatenbank-Eintrag gefunden. Wir bezeichnen ein Ergebnis auch dann als korrekt, wenn der Referenz in `references.matching.qrels` das Schlüsselwort `NO_MATCH` zugeordnet wurde und key_m keinen Wert enthält. Wie die Experimente zur Evaluation der Titel-Zuordnung führen wir diese Experimente einmal für $k = 50$ und einmal für $k = 500$ aus.

Ergebnisse & Diskussion

Korrektheitsanalyse *Referenzen-Zuordnung, k = 50*	**DBLP**	**Medline**
Anzahl zuzuordnender Referenzen	500	496
Korrekte Referenzen-Zuordnungen	450 90%	412 83,1%

Tabelle 5.11: Ergebnisse der Korrektheitsanalyse für die Referenzen-Zuordnung aus Publikationen von DBLP und Medline mit $k = 50$.

Insgesamt haben wir die Referenzen-Zuordnung an 500 Referenzen aus DBLP-Publikationen und 500 Referenzen aus Medline-Publikationen getestet. Für $k = 50$ konnten wir 90% der DBLP-Referenzen und 83,1% der Medline-Referenzen zuordnen (vgl. Tabelle 5.11). Neben den schon in den vorherigen Kapiteln erwähnten OCR-Dateien beeinflussen vor allem entweder Schreibfehler oder alternative Schreibweisen von Autorennamen oder Titeln das Ergebnis. Zusätzlich existierten von einigen Publikationen mehrere Versionen, die zwar von den selben Autoren und mit dem gleichen Titel, aber auf verschiedenen Konferenzen veröffentlicht wurden. Dadurch war es möglich, dass die verschiedenen Versionen verschiedene Erscheinungsjahre aufwiesen und somit verschiedene Bewertungen von unserem Algorithmus erhielten. Wenn zudem das Erscheinungsjahr in der zuzuordnenden Referenz entweder fehlerhaft[2] oder gar nicht aufgeführt war, wurde die Referenz unter Umständen einer falschen Version der Publikation zugeordnet.

Laufzeitanalyse *Referenzen-Zuordnung, k = 50*	**DBLP**	**Medline**
Gesamtlaufzeit, pro Referenz	4,32 ms	26,96 ms
Kandidaten finden	3,69 ms *85,4%*	24,47 ms *90,8%*
Kandidaten bewerten	0,62 ms *14,4%*	2,48 ms *9,2%*
1 Kandidaten auswählen	0,01 ms *0,2%*	0,01 ms *< 0,1%*

Tabelle 5.12: Detaillierte Ergebnisse der Laufzeitanalyse für die Referenzen-Zuordnung aus Publikationen von DBLP und Medline mit $k = 50$.

Für die Zuordnung einer DBLP-Referenz brauchte unser Algorithmus im Schnitt 4,32ms und für eine Medline-Referenz im Schnitt 26,96ms (vgl. Tabelle 5.12). Dabei konnte das Bewerten der Kandidaten deutlich schneller bewerkstelligt werden, als bei der Titel-Extraktion: Dauerte das Bewerten der Kandidaten während der Titel-Zuordnung noch 2,01ms pro DBLP-Titel bzw. 3,86 pro Medline-Titel, so benötigt die Referenzen-Zuordnung für diese Aufgabe nur 0,62ms pro DBLP-Referenz bzw. 2,48ms pro Medline-Referenz. Grund ist der von uns vorgeschaltete Autoren-Score, mit dem alle Kandidaten, die keinerlei Autorennamen mit der zuzuordnenden Referenz gemeinsam haben, herausgefiltert werden. Dadurch müssen wir die rechenintensive Berechnung der Ähnlichkeit zwischen Referenz und Kandidaten für deutlich weniger als 50 Kandidaten durchführen.

[2]Es kann vorkommen, dass sich das angegebene Jahr in der Referenz vom angegebenem Erscheinungsjahr in der Literaturdatenbank unterscheidet, wenn z.B. statt dem Erscheinungsjahr das Erstellungsjahr angegeben wird.

Korrektheitsanalyse *Referenzen-Zuordnung, k = 500*	**DBLP**	**Medline**
Anzahl zuzuordnender Referenzen	500	500
Korrekte Referenzen-Zuordnungen	474 *94,8%*	468 *93,6%*

Tabelle 5.13: Ergebnisse der Korrektheitsanalyse für die Referenzen-Zuordnung aus Publikationen von DBLP und Medline mit $k = 500$.

Erhöhen wir den Wert von k auf 500, erzielen wir ein Ergebnis von 94,8% korrekt zugeordneter DBLP-Referenzen sowie 93,6% korrekt zugeordneter Medline-Referenzen (vgl. Tabelle 5.13). Jedoch steigt damit auch hier wieder die Gesamtlaufzeit unseres Algorithmus deutlich an: Die Referenzen-Zuordnung dauert nun durchschnittlich 8,4ms pro DBLP-Referenz und 37,94ms pro Medline-Referenz (vgl. Tabelle 5.14).

Laufzeitanalyse *Referenzen-Zuordnung, k = 500*	**DBLP**	**Medline**
Gesamtlaufzeit, pro Referenz	8,4 ms	37,94 ms
Kandidaten finden	3,61 ms *43,0%*	24,37 ms *64,2%*
500 Kandidaten bewerten	4,78 ms *56,9%*	13,56 ms *35,7%*
1 Kandidaten auswählen	0,01 ms *0,1%*	0,01 ms *< 0,1%*

Tabelle 5.14: Detaillierte Ergebnisse der Laufzeitanalyse für die Referenzen-Zuordnung aus Publikationen von DBLP und Medline mit $k = 500$.

Damit steigt die Gesamtlaufzeit auf etwa das Doppelte an. Da wir durch die Erhöhung von k gleichzeitig die Anzahl der zu bewertenden Kandidaten erhöhen, steigt folgerichtig insbesondere die Laufzeit zur Bewertung der Kandidaten an. Wie schon für $k = 50$ hängt diese Erhöhung im Gegensatz zur Titel-Zuordnung aber nicht nur von k, sondern auch von der Anzahl an Kandidaten, die durch den Autoren-Score vor der Bewertung herausgefiltert werden, ab. Die Ergebnisse unserer Laufzeitanalyse ergaben, dass sich dadurch die Laufzeit nicht um den Faktor 10, sondern durchschnittlich nur noch etwa um den Faktor 7,5 erhöht. Die Erhöhung des Wertes k von 50 auf 500 lohnt sich deshalb mehr als bei der Titel-Zuordnung: Für den DBLP-Testdatensatz steigt mit der Erhöhung von k die Korrektheit der Zuordnungen um etwa 5,3% (von 450 auf 474 Referenzen), wobei die Laufzeit um etwa 94,4% steigt (von 4,32ms auf 8,4ms). Dagegen steigt die Zuordnungs-Korrektheit für den Medline-Datensatz um 13,59% (von 412 auf 468 Referenzen) und die Laufzeit um 40,73% (von 26,96ms auf 37,94ms).

Für eine sachgemäße Benutzung der Benutzerschnittstelle, die wir im nächsten Kapitel präsentieren werden, sind wir auf möglichst genaue Daten angewiesen. Wir werden für k deshalb den Wert 500 übernehmen, auch wenn wir dadurch Einbußen bei den Laufzeiten in Kauf nehmen.

5.7 Fazit

Wir haben gesehen, dass die Ergebnisse und die Effizienz sowohl der Extraktion als auch der Zuordnung von Titeln und Referenzen wissenschaftlicher Publikationen von zahlreichen Faktoren abhängen. Für eine korrekte Extraktion von Titel und Referenzen sind wir besonders auf die Korrektheit der von der Java-Bibliothek *PDFBox* gelieferten Angaben und Inhalte angewiesen, die vor allem bei sogenannten *OCR-Dateien* nicht immer gegeben war. Auch die Effizienz der Extraktion hängt stark von PDFBox ab, da ihre Aufgaben den Hauptanteil der Laufzeiten ausmachen.

Dank der zusätzlichen Zuordnung sind aber Fehler bei der Extraktion vertretbar, solange der repräsentierende Eintrag in der Literaturdatenbank trotzdem gefunden wird und wir somit die korrekten Metadaten erhalten. Dabei hängt sowohl die Korrektheit als auch die Effizienz der Zuordnung stark von der Wahl des Wertes für die Anzahl k der zu bewertenden

Kandidaten ab. Wie bereits diskutiert, ist dieser Wert sorgfältig auszuwählen, um ein angemessenes Gleichgewicht zwischen Korrektheit und Effizienz der Zuordnung zu erhalten.

Die ermittelten Ergebnisse der Korrektheits-Analysen sind zu den in Kapitel 2 vorgestellten Ergebnissen vergleichbar, auch wenn wir die Metadaten der Publikationen und der Referenzen nicht ganz mit den Genauigkeiten, wie sie von Methoden des Maschinellen Lernens erreicht werden, identifizieren können. Leider konnten wir bei unserer Literaturrecherche so gut wie keine Angaben zu Laufzeiten der Methoden des Maschinellen Lernens finden, weshalb wir nicht beurteilen können, wie effizient die in Kapitel 2 genannten Ergebnisse mit *Hidden Markov Modellen*, *Support Vector Machines* und *Conditional Random Fields* berechnet werden können. Lediglich Jöran Beel et al. nennen für die Titel-Extraktion mit einer regelbasierten Heuristik eine durchschnittliche Laufzeit von 0,72 Sekunden pro PDF-Datei (8 Minuten und 19 Sekunden für 693 PDF-Dateien). Diese Laufzeit unterbieten wir mit durchschnittlich rund 43 Millisekunden (0,043 Sekunden) pro PDF-Datei deutlich.

Kapitel 6

Die Benutzerschnittstelle

Die in den Kapiteln 3 und 4 vorgestellten Algorithmen zur Extraktion und Zuordnung von Titeln und Referenzen wissenschaftlicher Publikationen werden wir nun in eine Benutzerschnittstelle integrieren, so dass sie allgemein benutzbar und für Benutzer zugänglich werden. Mit der Schnittstelle können Benutzer eine persönliche *Bibliothek* wissenschaftlicher Publikationen verwalten. Die Publikationen können als PDF-Dateien in die Bibliothek geladen werden, aus denen das System Titel und Referenzen extrahiert und Einträgen aus DBLP oder Medline zuordnet. Die daraus erhaltenen Metadaten werden dann dazu verwendet, die Publikationen in der Bibliothek möglichst genau zu beschreiben, so dass der Benutzer sie eindeutig identifizieren kann. Zudem kann er auf sehr einfache Weise verwandte Publikationen finden und ebenfalls seiner Bibliothek hinzufügen.

6.1 Web- oder Desktop-Anwendung?

Vor der Implementierung der Benutzerschnittstelle stellte sich für uns die Frage, ob diese als eine Web- oder als eine Desktop-Anwendung realisiert werden sollte. Um eine Entscheidung treffen zu können, haben wir Vor- und Nachteile beider Varianten untersucht:

- **Web-Anwendung:** Eine Web-Anwendung bietet den großen Vorteil der Mobilität. Im Gegensatz zur einer Desktop-Anwendung können Web-Anwendungen mit jedem beliebigen Endgerät mit Internetzugang und geeignetem Webbrowser benutzt werden. Diese Eigenschaft machen Web-Anwendungen zusätzlich plattformunabhängig, da Webbrowser für alle gängigen Betriebssysteme existieren. Eine Web-Anwendung ist von einem zentralen Server abrufbar. Hierzu sind keine clientseitigen Installationen notwendig. Dadurch sind auch mögliche Updates an zentraler Stelle einspielbar und müssen nicht von jedem Client einzeln installiert werden. Allerdings ist man bei einer Web-Anwendung von der Verfügbarkeit des Servers abhängig. Sollte dieser einmal nicht erreichbar sein, so hat man keinerlei Zugriff auf seine Daten. Zusätzlich müssen alle Daten und Dateien zunächst entweder vom Client zum Server oder vom Server zum Client übertragen werden. Je nach Geschwindigkeit der Internetverbindung nimmt dies unterschiedlich viel Zeit in Anspruch und wiegt dieser Nachteil unterschiedlich schwer.

- **Desktop-Anwendung:** Da Desktop-Anwendungen lokal installiert werden, sind alle Nachteile einer Web-Anwendung zugleich Vorteile einer Desktop-Anwendung: Alle Funktionalitäten und alle Daten sind lokal vorhanden und müssen nicht zu einem Server gesendet werden. Wir sind deshalb zwar nicht auf die Verfügbarkeit eines Servers angewiesen, dafür aber von der lokalen Installation abhängig. Alle gespeicherten Daten und Dateien sind nur von dieser Installation abrufbar. Zwar wären mehrere Installationen auf verschiedenen Rechnern denkbar, das Problem ist dann aber, dass alle Installationen auf ihren eigenen Datenbeständen arbeiten würden. Möchte man die Datenbestände synchron halten, wäre zusätzlicher Aufwand notwendig. Zudem machen verschiedene Betriebssysteme und Systemvoraussetzungen die Entwicklung einer Desktop-

Anwendung wesentlich komplexer als die Entwicklung einer Web-Anwendung. Möchte man eine weitgehende Plattformunabhängigkeit erreichen, müssten für alle zu unterstützenden Betriebssysteme unterschiedliche Versionen implementiert werden.

Nach Abwägung der genannten Vor- und Nachteile haben wir uns dazu entschlossen, die Benutzerschnittstelle als Web-Anwendung zu implementieren. Ausschlaggebend für unsere Entscheidung war die Tatsache, dass die oben dargelegten Vorteile einer Web-Anwendung gegenüber den Vorteilen einer Desktop-Anwendung deutlich überlegen waren. Im Hinblick auf den jetzt schon allgemein hohen Stellenwert des Internets im Alltag und der immer größer werdenden Verfügbarkeit von Breitbandverbindungen verlieren die genannten Nachteile einer Web-Anwendung zudem immer mehr an Bedeutung.

6.2 Die Anwendungsumgebung

Die allgemeine Architektur der Web-Anwendung folgt dem üblichen *Client-Server-Modell* (vgl. Abbildung 6.1): Ein zentraler *Server* stellt die Dienste des gesamten Systems zur Verfügung, die von *Clients* angefordert werden können. Der Server der Anwendung kommuniziert selbst

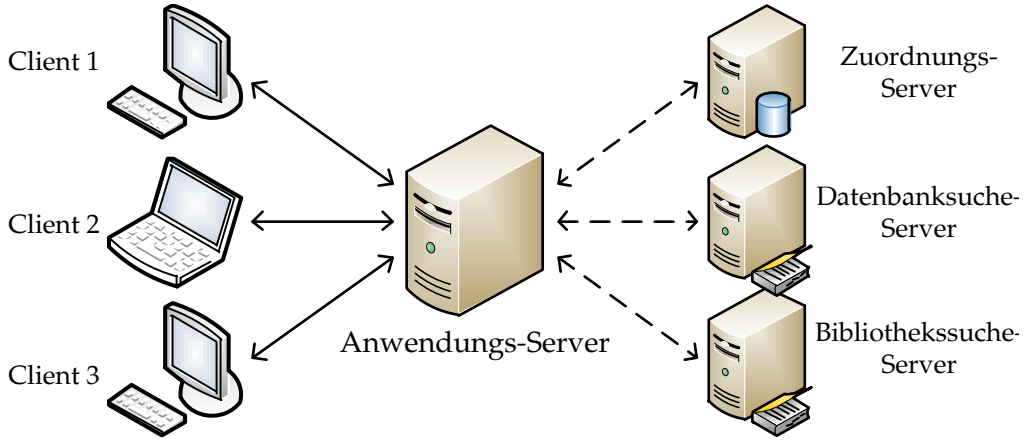


Abbildung 6.1: Die Anwendungsarchitektur folgt dem üblichen Client-Server-Modell: *Clients* fordern die vom *Anwendungs-Server* zur Verfügung gestellten Dienste an. Der Anwendungs-Server kommuniziert dazu mit dem *Zuordnungs-Server*, dem *Datenbanksuche-Server* und dem *Bibliothekssuche-Server*.

wiederum mit drei weiteren Servern, nämlich mit dem Zuordnungs-Server (*kurz*: Z-Server), dem Datenbanksuche-Server (DS-Server) und dem Bibliothekssuche-Server (BS-Server):

- **Z-Server:** Der Z-Server ist ein HTTP-Server und wartet an einem definierten Port auf Zuordnungs-Anfragen. Diese Anfragen ordnet der Server einem Eintrag aus DBLP oder Medline zu. Er führt also genau die in Kapitel 4 vorgestellten Algorithmen aus. Obwohl diese in C++ implementiert wurden, kann der Anwendungs-Server, dessen Quellcode in Java geschrieben wurde (siehe unten), dank dieser Serverschnittstelle mit der Zuordnungs-Komponente kommunizieren. Um Einträge sowohl aus DBLP als auch aus Medline zu finden, wird der zugrunde liegende Index des Z-Servers aus der XML-Datei `dblp+medline.xml` erstellt, in der alle Einträge aus `dblp.xml` und `medline.xml` zusammengeführt wurden. Je nachdem, ob die Zuordnungsanfrage mit dem Parameter `"t"` oder mit dem Parameter `"r"` gestellt wird, führt der Server entweder eine Titel-Zuordnung oder eine Referenzen-Zuordnung durch (Beispiel 6.2). Da wir den für die Zuordnung benötigten Index nicht über alle verfügbaren Metadaten der Literaturdatenbanken erstellt haben, können wir von ihm nicht alle notwendigen Metadaten erhalten. Der Z-Server antwortet auf eine Suchanfrage deshalb mit einem XML-Dokument, dass lediglich die Keys der Zuordnungs-Kandidaten, absteigend sortiert nach ihrem Bewertungsscore, enthält (Beispiel 6.3).

```
http://<Host>:<Port>/?t=some+query
```

Beispiel 6.2: Die Syntax einer Zuordnungsanfrage (hier: *»some query«*) an den Zuordnungs-Server. Da die Anfrage mit dem Parameter `"t"` gestellt wird, handelt es sich um eine Anfrage für eine Titel-Zuordnung.

```
<response query="some query">
  <inproceedings key="conf/er/CatarciS88"/>
  <inproceedings key="conf/airs/SunLZ08"/>
  <inproceedings key="conf/edbt/PentarisI04"/>
  <inproceedings key="conf/edbtw/Dekeyser02"/>
  <inproceedings key="conf/edbt/Dekeyser02"/>
  <inproceedings key="conf/acl/KatragaddaV09"/>
  <inproceedings key="conf/esws/HartigH07"/>
  <inproceedings key="conf/edbtw/Baeza-YatesHM04"/>
  <inproceedings key="conf/cikm/BaragliaCDNPS09"/>
  <inproceedings key="conf/aina/MakkiR10"/>
  <inproceedings key="conf/fqas/ColletV04"/>
  <inproceedings key="conf/dbsec/PalazziPP10"/>
  <inproceedings key="conf/edbt/NehmeRB09"/>
  <inproceedings key="conf/dexa/WangA10"/>
  [...]
</response>
```

Beispiel 6.3: XML-Antwort des Zuordnungs-Servers, die die Keys der Zuordnungs-Kandidaten der Titel-Zuordnung für die Anfrage *»some query«* enthält. Die Keys sind absteigend nach ihrem Bewertungsscore, den sie während der Zuordnung erhalten haben, sortiert.

- **DS-Server:** Die vollständigen Metadaten zu einem Key erhalten wir über Anfragen an den DS-Server, mit dem wir DBLP und Medline durchsuchen können. Ebenso wie der Z-Server ist der DS-Server ein HTTP-Server und wartet an einem definierten Port auf Suchanfragen. Das Backend dieses Servers basiert auf der von Hannah Bast und Ingmar Weber entwickelten Suchmaschine *CompleteSearch* (siehe Kapitel 6.2.1), die eine schnelle und effiziente Suche in großen Datenbeständen ermöglicht. Der zugrunde liegende Index wird ebenfalls über die XML-Datei `dblp+medline.xml` erstellt. Um zu einem Key die vollständigen Metadaten zu erhalten, können wir eine Suchanfrage der Form, wie sie in Beispiel 6.4 zu sehen ist, an den DS-Server stellen. Dieser antwortet wieder mit einem XML-Dokument, das die gewünschten Metadaten enthält, sofern ein Eintrag mit diesem Key existiert (vgl. Beispiel 6.5).

```
http://<Host>:<Port>/?q=ce:key:conf_naacl_pengm04*
```

Beispiel 6.4: Suchanfrage an den DS-Server, um für den DBLP-Eintrag mit dem Key `conf/naacl/PengM04` die vollständigen Metadaten zu erhalten.

- **BS-Server:** Mithilfe des BS-Servers kann ein Benutzer seine persönliche Bibliothek durchsuchen. Durchsuchbar sind sämtliche Metadaten der in der Bibliothek enthaltenen wissenschaftlichen Publikationen, inklusive der Volltexte aus den PDF-Dateien. Da die zu durchsuchende Bibliothek für jeden Benutzer individuell ist, benötigen wir für jeden angemeldeten Benutzer einen individuellen BS-Server. Eine besondere Herausforderung ist zudem, den Index des BS-Servers stets aktuell zu halten, da sich der Datenbestand einer Benutzer-Bibliothek durch das Hinzufügen, Editieren oder Löschen von Metadaten wissenschaftlicher Publikationen ständig verändert. Wie der DS-Server basiert ein BS-Server auf *CompleteSearch*, weshalb die Suchanfragen und XML-Antworten identisch zu den Beispielen 6.4 und 6.5 aufgebaut sind. Alle Details zur Verwaltung der individuellen BS-Server werden wir später im Kapitel zur Suche vorstellen.

Um die Antwortzeiten der jeweiligen Server so gering wie möglich zu halten, sind möglichst schnelle Anbindungen an die drei Server zu garantieren. Idealerweise laufen alle Server auf

```
<result>
  <query>ct:key:confnaaclpengm04*</query>
  <status code="12">OK</status>
  <time unit="msecs">741.37</time>
  <completions total="1" computed="1" sent="1">
    <c sc="2" dc="1" oc="1" id="13495522">ct:key:confnaaclpengm04:co[...]
  </completions>
  <hits total="1" computed="1" sent="1" first="0">
    <hit score="2" id="1753747">
      <title>
        <![CDATA[
          <dblp:authors>
            <dblp:author>Fuchun Peng</dblp:author>
            <dblp:author>Andrew McCallum</dblp:author>
          </dblp:authors>
          <dblp:title ee="http://acl.ldc.upenn.edu/hlt-naacl2004/m[...]">
            Accurate Information Extraction from Research Papers usi[...]
          </dblp:title>
          <dblp:venue url="db/conf/naacl/naacl2004.html#PengM04">
            HLT-NAACL 2004:329-336
          </dblp:venue>
          <dblp:year>2004</dblp:year>
          <dblp:type>inproceedings</dblp:type>
        ]]>
      </title>
      <url>
        http://dblp.uni-trier.de/rec/bibtex/conf/naacl/PengM04
      </url>
    </hit>
  </hits>
</result>
```

Beispiel 6.5: XML-Antwort des DS-Servers, die die vollständigen Metadaten (Autoren, Titel, Erscheinungsjahr, etc.) des Literaturdatenbank-Eintrags. mit dem Key `conf/naacl/PengM04` enthält.

dem gleichen Rechner, zumindest aber auf Rechnern, die sich im selben (lokalen) Netzwerk befinden.

6.2.1 CompleteSearch

Mit der von Hannah Bast und Ingmar Weber entwickelten Suchmaschine *CompleteSearch* [27] sind insbesondere sogenannte *Präfix-Suchen* effizient durchführbar. Bei einer Präfix-Suche werden für ein Wort w der Suchanfrage zusätzlich alle Wörter berücksichtigt, die w als Präfix enthalten. Eine Präfix-Suche mit der Suchanfrage `inf*` findet alle Dokumente, die z.B. das Wort `inflation`, `influenza` oder `informatics` enthalten. Mit einem Invertierten Index müssten für eine Präfix-Suche die invertierten Listen aller Vervollständigungen des Wortes `inf` aus dem Index bestimmt und vereinigt werden. Wie wir aber schon von der Evaluation der Titel- und Referenzen-Zuordnung aus Kapitel 5 wissen, ist die Vereinigung von invertierten Listen eine vergleichsweise zeitaufwändige Operation.

CompleteSearch verwendet deshalb eine Weiterentwicklung des Invertierten Index namens *HYB* [26], die eine Vereinigung der invertierten Listen überflüssig macht. Die Idee hinter HYB ist die Vorausberechnung von invertierten Listen für die Vereinigung von Wörtern aus einem definierten Wortbereich (*Block*). Anstatt für jedes zu indizierende Wort werden die invertierten Listen in HYB für jeden Block berechnet, was im folgenden Beispiel an den Wörtern A, B, C, D, E und F gezeigt sei:

$$\begin{aligned}
\texttt{A} &\mapsto 1, 4\\
\texttt{B} &\mapsto 3, 4, 7, 12\\
\texttt{C} &\mapsto 2, 12, 17\\
\texttt{D} &\mapsto 11, 17, 18\\
\texttt{E} &\mapsto 1, 8, 9, 13, 15\\
\texttt{F} &\mapsto 8
\end{aligned}$$

Obiger Ausschnitt zeigt die herkömmliche Methode eines Invertierten Index, die Vorkommen von Wörtern in Dokumenten zu speichern. Der HYB-Index berechnet dagegen die invertierten Listen für Blöcke von Wörtern:

Block A-C	1	2	3	4	4	7	12	12	17
	A	C	B	A	B	B	B	C	C
Block D-F	1	8	8	9	11	13	15	17	18
	E	E	F	E	D	E	E	D	D

Die Elemente der invertierten Listen aus HYB sind Tupel (i, w), in denen zu jeder ID i zusätzlich das Wort w, das zu der Einteilung in den entsprechenden Block geführt hat, gespeichert wird. Mit HYB werden bei einer Präfix-Suche die zeitaufwändigen Vereinigungen der invertierten Listen vermieden, weil nach Konstruktion alle möglichen Vervollständigungen eines Wortes in einem Block liegen. Bei einer Suchanfrage muss deshalb immer ein kompletter Block gelesen werden. Da dieser auch zur Suchanfrage irrelevante Dokumente enthalten kann, müssen alle irrelevanten Tupel (i, w) jeweils anhand des zusätzlich gespeicherten Wortes w aus dem Block herausgefiltert werden. Alle Details, u.a. wie durch geschickte Kodierungen und Komprimierungen der invertierten Listen die Performanz von HYB optimiert wird, sind den Ausführungen von Bast und Weber [26] zu entnehmen.

Außerdem können wir mit CompleteSearch eine strukturierte Suche in XML-Dokumenten durchführen, d.h. eine Suche, die nur Werte definierter XML-Elemente berücksichtigt. Dazu werden allen Wörtern bei der Indizierung ihre XML-Tags zusammen mit einem beliebigen Namespace (wir verwenden den Namespace `ce`) vorangestellt und als Spezialwörter in das Vokabular der Suchmaschine eingefügt. Beispielsweise wird das Wort `extraction` aus dem XML-Element `<title>` sowohl mit `extraction` als auch mit `ce:title:extraction` indiziert. Eine Präfix-Suche `ce:title:ex*` liefert dann alle Dokumente, die im Titel mindestens ein Wort enthalten, das mit `ex` beginnt.

Mit der Suchanfrage q = `ce:key:conf_naacl_pengm04*` aus Beispiel 6.4 suchen wir in `dblp+medline.xml` nach einem Eintrag, der den Key `conf/naacl/PengM04` besitzt. Das Zeichen `"/"` wird durch das Zeichen `"_"` ersetzt, um mögliche Konflikte mit *Slashes* aus URL's zu vermeiden. Speziell bei der Suche nach Keys ist eine sorgfältige Wahl der Blockgrößen notwendig. Ein Block `ce:key:` würde zum Beispiel eine Länge in der Anzahl der Literaturdatenbank-Einträge besitzen. Obwohl wir mit q nur einen einzigen Eintrag suchen, müsste der Block komplett gelesen und alle irrelevanten Tupel (also alle bis auf einen) herausgefiltert werden. Im Falle von `dblp+medline.xml` wären das über 20 Millionen solcher Tupel, die gelesen werden müssten, um einen Eintrag zu finden. Es würden nicht vertretbare Antwortzeiten für derartige Suchanfragen resultieren. Abhilfe schafft eine Verkleinerung der Blöcke, um für die Suchanfragen insgesamt weniger Daten lesen zu müssen. Allerdings sind die Blockgrößen wiederum nicht zu klein zu wählen, um eventuell notwendige Vereinigungen von invertierten Listen zu vermeiden. Wir wählen eine Einteilung in die Blöcke `ce:key:conf`, `ce:key:books`, etc.

6.2.2 Datenverwaltung

```
uploads/
  <user>/
    tmp/
    xml/
    conf.ksem.GaoVZ07.pdf
    conf.ksem.GaoVZ07_annot.pdf
    conf.iros.MosteoM09.pdf
    conf.iros.MosteoM09_annot.pdf
    library.xml
```

Beispiel 6.6: Verzeichnisstruktur, in der die Dateien und Daten eines Benutzers gespeichert werden. Für jeden Benutzer wird ein individueller Ordner angelegt, der die zwei Ordner `tmp/` und `xml/` sowie alle PDF-Dateien und die Bibliotheksdatei `library.xml` enthält.

Alle Dateien und Daten eines Benutzers werden in einer Verzeichnisstruktur gespeichert, wie sie in Beispiel 6.6 zu sehen ist. Jeder Benutzer der Anwendung erhält einen individuellen Ordner ∼/uploads/<user> („∼“ bezeichne ein beliebiges, aber fest definiertes Wurzelverzeichnis für alle zu speichernden Daten der Anwendung und <user> den Namen des Benutzers). Alle Metadaten, die wir durch die Extraktion und Zuordnung von Titel und Referenzen erhalten, speichern wir in der Bibliotheksdatei library.xml ab. Wir verzichten hier auf ein relationales Datenbanksystem und speichern die Daten in XML-Dateien ab, um Homologien bei der Indexierung der Suchmaschinen zu erreichen (denn die Daten aus DBLP und Medline liegen ebenfalls als XML-Datei vor). Die Syntax ist stark an die von dblp.xml und medline.xml angelehnt und wurde von uns lediglich um folgende Elemente erweitert:

- <cite>: In diesem Element speichern wir eine Referenz einer Publikation. Jedes <cite>-Element enthält mit <citekey>, <citetitle>, <citeauthor>, <citeyear>, <citeee>, <citeurl> und <citevenue> alle bereits bekannten Elemente eines Literaturdatenbank-Eintrags. Zusätzlich speichern wir in <citeextract> die extrahierte Zeichenkette einer Referenz.
- <fulltext>: Dieses Element speichert den extrahierten Volltext einer Publikation.
- <filename>: Dieses Element speichert den Dateinamen der PDF-Datei einer Publikation.
- <annotcopy>: Dieses Element speichert den Dateinamen der *annotierten* PDF-Datei einer Publikation. In einer annotierten PDF-Datei erhält jede identifizierte Referenz eine Annotation, die z.B. eine individuelle URL enthalten kann. Alle Details zu annotierten PDF-Dateien werden wir zu einem späteren Zeitpunkt vorstellen.

Beispiel 6.7 zeigt einen Ausschnitt einer library.xml. Für die Indexierung von library.xml durch den BS-Server können wir die bereits zur Indexierung von dblp.xml und medline.xml geschaffene Infrastruktur (wie z.B. den XML-Parser) nutzen.

Der Ordner des Benutzers enthält zudem einen Ordner tmp/, in dem alle temporär benötigten Dateien (siehe unten) gespeichert werden, und einen Ordner xml/, dessen Zweck wir im folgenden Abschnitt erklären.

Manipulation von library.xml

Wie wir bereits wissen, speichern wir alle Metadaten der Bibliothek in library.xml ab. Zu jeder wissenschaftlichen Publikation der Bibliothek speichern wir sowohl ihre Metadaten inklusive des Volltextes, als auch die Metadaten zu jeder ihrer Referenzen. Deshalb kann library.xml beliebig groß werden. Weil ein Benutzer wissenschaftliche Publikationen hinzufügen und löschen, sowie Metadaten editieren können soll, müssen wir aber in der Lage sein, library.xml möglichst effizient zu manipulieren, d.h. XML-Elemente hinzuzufügen, zu editieren und zu löschen. Ein allgemeiner Ansatz zur Manipulation von XML-Dateien ist DOM (*Document Object Model*, [24]). DOM liest die gesamte zu manipulierende XML-Datei in den Arbeitsspeicher und erstellt zudem zusätzliche Datenstrukturen, um gewünschte XML-Elemente ansteuern und modifizieren zu können. Aus diesen Gründen hat DOM einen enormen Speicherbedarf und besitzt eine vergleichsweise niedrige Performanz. DOM kommt zur Manipulation von library.xml deshalb für uns nicht in Frage.

Ein weiterer Ansatz zur Manipulation der XML-Datei library.xml wäre ihre komplette Neuerstellung bei jeder Änderung. Obwohl eine Änderung nur einen kleinen Teil der XML-Datei betreffen kann, müsste library.xml hierbei komplett neu geschrieben werden. Angesichts der möglichen Dateigröße von library.xml ist auch dieser Ansatz nicht effizient.

Schreiben von library.xml Stattdessen erstellen wir für jede wissenschaftliche Publikation eine sogenannte XML-Fragment-Datei, die genau das XML-Fragment enthält, das die Daten der Publikation enthält. Jede dieser Fragment-Dateien speichern wir im Ordner xml/ ab und benennen sie entsprechend des Keys der Publikation nach dem Muster in Beispiel 6.8. Die XML-Datei library.xml erhalten wir dann, indem wir alle Fragment-Dateien mit der Befehlssequenz aus

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <article filename="[...]" key="conf/naacl/PengM04" annotcopy="[...]">
    <title>Accurate Information Extraction from Research Papers usin[...]</title>
    <author>Fuchun Peng</author>
    <author>Andrew McCallum</author>
    <year>1999</year>
    <fulltext><![CDATA[Accurate Information Extraction from Re[...]]]></fulltext>
    <ee>http://acl.ldc.upenn.edu/hlt-naacl2004/main/pdf/176_Paper.pdf</ee>
    <url>http://dblp.uni-trier.de/rec/bibtex/conf/naacl/PengM04</url>
    <venue>HLT-NAACL 2004:329-336</venue>
    <cite>
       <citeextract>S. Chen and R. Rosenfeld. 2000. A Survey o[...]</citeextract>
    </cite>
    <cite citekey="conf/naacl/Goodman04">
      <citetitle>Exponential Priors for Maximum Entropy Models.</citetitle>
      <citeauthor>Joshua Goodman</citeauthor>
      <citeyear>2004</citeyear>
      <citeee>http://acl.ldc.upenn.edu/hlt-naacl2004/main/pdf/22_Pa[...]</citeee>
      <citeurl>http://dblp.uni-trier.de/rec/bibtex/conf/naacl/Good[...]</citeurl>
      <citevenue>HLT-NAACL 2004:305-312</citevenue>
      <citeextract>J. Goodman. 2003. Exponential Priors for Ma[...]</citeextract>
    </cite>
    <cite citekey="conf/jcdl/HanGMZZF03">
      <citetitle>Automatic Document Metadata Extraction Using Su[...]</citetitle>
      <citeauthor>Hui Han</citeauthor>
      <citeauthor>C. Lee Giles</citeauthor>
      <citeauthor>Eren Manavoglu</citeauthor>
      <citeauthor>Hongyuan Zha</citeauthor>
      <citeauthor>Zhenyue Zhang</citeauthor>
      <citeauthor>Edward A. Fox</citeauthor>
      <citeyear>2003</citeyear>
      <citeee>http://csdl.computer.org/comp/proceedings/jcddl/2003/[...]</citeee>
      <citeurl>http://dblp.uni-trier.de/rec/bibtex/conf/jcdl/HanGM[...]</citeurl>
      <citevenue>JCDL 2003:37-48</citevenue>
      <citeextract>H. Han, C. Giles, E. Manavoglu, H. Zha, Z. [...]</citeextract>
    </cite>
    [...]
  </article>
  [...]
</library>
```

Beispiel 6.7: Ausschnitt aus einer Bibliotheksdatei `library.xml`, in der die aus der Titel- und Referenzen-Zuordnung resultierenden Metadaten für Publikationen und Referenzen gespeichert werden.

Beispiel 6.9 zusammensetzen: Der Befehl in Zeile 1 fügt die Definition, welche XML-Version und welche Zeichenkodierung verwendet wird, als erste Zeile in `library.xml` ein. Die Befehle in Zeile 2 und Zeile 6 sorgen dafür, dass ein wohlgeformtes Wurzelelement `library` existiert. Die Schleife in den Zeilen 3-5 verknüpft mit dem Linux-Befehl `cat` alle Dateien aus ~/uploads/<user>/xml, aufsteigend sortiert nach ihrem Änderungsdatum, und speichert sie in `library.xml` ab.

Änderungen in einer Bibliothek können wir nun sehr einfach und effizient bewerkstelligen: Für das Hinzufügen einer Publikation müssen wir lediglich eine neue Fragment-Datei erstellen. Beim Editieren von Metadaten können wir dank des gewählten Musters für die Benennung der Fragment-Dateien die zu editierende Fragment-Datei anhand des Keys identifizieren. Wir verzichten darauf, die Änderungen von Metadaten aufwändig in die Fragment-Datei einzupflegen und schreiben die Fragment-Datei stattdessen komplett neu. Das Löschen einer Publikation können wir durch das Löschen der entsprechenden Fragment-Datei bewerkstelligen. Nach jeder Änderung setzen wir aus allen in ~/uploads/<user>/xml/ verfügbaren Fragment-Dateien die XML-Datei `library.xml` neu zusammen, so dass sie vom System verwendet und vom BS-Server indiziert werden kann.

Festzuhalten ist, dass wir dank der Modularisierung von `library.xml` immer nur einen kleinen Teil der XML-Daten bearbeiten müssen. Wir bearbeiten in keinem Fall die komplette

Key der Publikation:	`conf/naacl/PengM04`
Dateiname der XML-Fragment-Datei:	`conf.naacl.PengM04.part`

Beispiel 6.8: Das Muster für den Dateinamen einer XML-Fragment-Datei. Der Dateiname entsteht aus dem Key der Publikation, indem das Zeichen `"/"` durch das Zeichen `"."` ersetzt wird und die Dateiendung `".part"` angehängt wird.

```
1 echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" > library.xml ; \
2 echo "<library>" >> library.xml ; \
3 for file in $$(ls -rt ~/uploads/<user>/xml) ; do \
4    cat $$file >> library.xml ; \
5 done ; \
6 echo "</library>" >> library.xml
```

Beispiel 6.9: Befehlssequenz, um aus den XML-Fragment-Dateien aus `~/uploads/<user>/xml/` die vollständige Bibliotheksdatei `library.xml` zusammenzusetzen.

Datei, was eine deutlich geringere Laufzeit und einen niedrigeren Speicherverbrauch bedeutet.

Lesen von `library.xml` Sowohl beim Auslesen der Daten aus `library.xml`, als auch bei der Auswertung der Antworten der verschiedenen Server, müssen wir jeweils XML-Dokumente parsen. Auch hierfür könnten wir wieder *DOM* verwenden, was wir aber aus den bereits bekannten Gründen unterlassen. Wir verwenden stattdessen einen sogenannten *SAX-Parser* (Simple API for XML, [20]). Ein SAX-Parser liest XML-Dateien sequentiell und arbeitet eventbasiert, d.h. bei jedem Ereignis (Ereignisse sind z.B. ein öffnendes XML-Element, ein schließendes XML-Element oder Text innerhalb eines XML-Elements) ruft der Parser eine definierte *callback*-Funktion auf. Durch die Definition dieser *callback*-Funktionen können wir auf diese Ereignisse individuell reagieren und die gewünschten Daten auslesen.

Ein SAX-Parser „kennt“ nur die jeweils aktuell betrachtete Zeile, speichert also keine Informationen über ihren Kontext und über zuvor besuchte Zeilen. Zwar müssen wir dadurch benötigte Informationen über den Kontext (z.B. zu welcher Publikation die aktuell betrachtete Zeile gehört) eigenständig zwischenspeichern, was die Verwendung von SAX-Parsern im Vergleich zu DOM-Parsern ein Stück weit aufwändiger macht. Gleichzeitig ist dadurch aber ein effizientes Parsen besonders von großen XML-Dateien möglich.

6.3 Die Implementierung der Anwendung

Bei der Implementierung der Anwendung diente uns als Framework das *Google Web Toolkit* (GWT) [11], das vor allem für das Erstellen und Optimieren von browserbasierten Applikationen entwickelt wurde. Wir haben uns für GWT als Framework entschieden, weil es den den großen Vorteil bietet, dass sowohl die Server- als auch die Client-Komponente komplett in Java geschrieben werden kann. Ein integrierter *Java-nach-Javascript-Compiler* übersetzt den Java-Code für die Client-Komponente in Javascript, HTML und CSS (vgl. schematische Darstellung des Prinzips von GWT in Abbildung 6.10). Dabei werden automatisch für jeden gängigen Webbrowser (z.B. Mozilla Firefox, Google Chrome, Internet Explorer, Apple Safari und Opera) optimierte Versionen erstellt, weshalb die Entwicklung von Web-Anwendungen mit GWT sehr komfortabel möglich ist.

Um den Code modular und damit wartbar sowie erweiterbar zu gestalten, folgt der Aufbau unseres Codes dem Entwurfsmuster *Model-View-Presenter* (MVP). Der Ansatz hierbei ist, den Code für die grafische Benutzeroberfläche (View) und den Code für die Logik (Presenter), die das Verhalten der Anwendung steuert und mit Daten (Model) bestückt, voneinander zu trennen (vgl. Abbildung 6.11).

Alle drei Komponenten wissen nichts über die Existenz der jeweils anderen Komponenten. Über definierte Schnittstellen können die Komponenten über bestimmte Ereignisse (z.B.

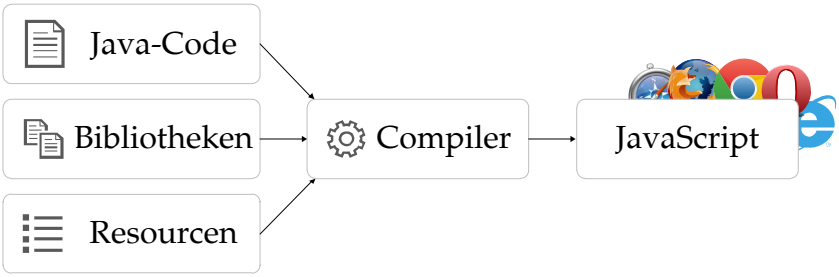


Abbildung 6.10: Schematische Funktionsweise des Java-nach-Javascript-Compiler von GWT. Der Java-Code und externe Bibliotheken sowie Ressourcen (wie z.B. Bilder) werden vom Compiler in entsprechenden JavaScript- sowie CSS-Code übersetzt und in eine HTML-Seite eingebettet. Dabei werden für jeden gängigen Webbrowser optimierte Versionen erstellt, um eine möglichst hohe Browserkompatibilität zu erreichen.

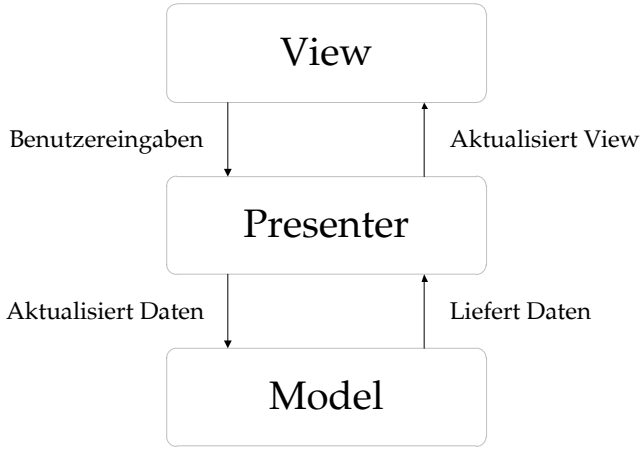


Abbildung 6.11: Schematische Darstellung des Entwurfsmusters *Model-View-Presenter*. Jede Komponente arbeitet eigenständig und besitzt keinerlei Wissen über die jeweils anderen Komponenten. Über definierte Schnittstellen können die Komponenten Informationen aussenden, auf die die restlichen Komponenten reagieren können.

Benutzereingaben) informieren. Wie die einzelnen Komponenten auf diese Informationen konkret reagieren, bleibt ihnen komplett selbst überlassen.

Dieses Entwurfsmuster macht die Komponenten beliebig austauschbar: So kann z.B. die Implementierung eines Elements der Benutzeroberfläche in der View-Komponente durch eine alternative Implementierung ausgetauscht werden, ohne dabei Änderungen an der Presenter-Komponente vornehmen zu müssen.

6.4 Die Anwendung aus Client-Sicht

Grundsätzlich ist die Web-Anwendung in drei sogenannte Sichten unterteilt: in die Login-Sicht, die Bibliothekssicht und in die Detailsicht.

6.4.1 Die Login-Sicht

Weil ein Benutzer seine eigene, individuelle Bibliothek verwalten können soll, müssen wir in der Lage sein, den Benutzer zu identifizieren. Für die notwendige Benutzerverwaltung nutzen wir die Infrastruktur der Technischen Fakultät der Universität Freiburg, so dass sich alle

Mitglieder der Technischen Fakultät mit den Anmeldedaten ihres Benutzerkontos, in unserer Anwendung anmelden können. Wir überprüfen die Korrektheit der Anmeldedaten, indem wir mit den eingegebenen Anmeldedaten eine SSH-Verbindung zu einem Rechner der Fakultät (`login.informatik.uni-freiburg.de`) aufbauen. Wird diese SSH-Verbindung erfolgreich aufgebaut, sind die Anmeldedaten korrekt und ihm wird der Zugang zur Anwendung gewährt, wo ihm seine persönliche *Bibliothekssicht* angezeigt wird. Anderenfalls konnten wir die Anmeldedaten nicht validieren, weshalb wir dem Benutzer den Zugang zur Anwendung nicht gestatten.

6.4.2 Die Bibliothekssicht

Die Bibliothekssicht ist die Hauptsicht der Anwendung. Hier werden u.a. alle Publikationen der Bibliothek eines Benutzers aufgelistet. Wie in Abbildung 6.12 zu sehen, unterteilt sich die Bibliothekssicht in 5 Bereiche: in die Kopfzeile (Bereich 1), in die Werkzeugleiste (Bereich 2), in die Publikationsliste (Bereich 3), in die Referenzenliste (Bereich 4) und in die Fußzeile (Bereich 5).

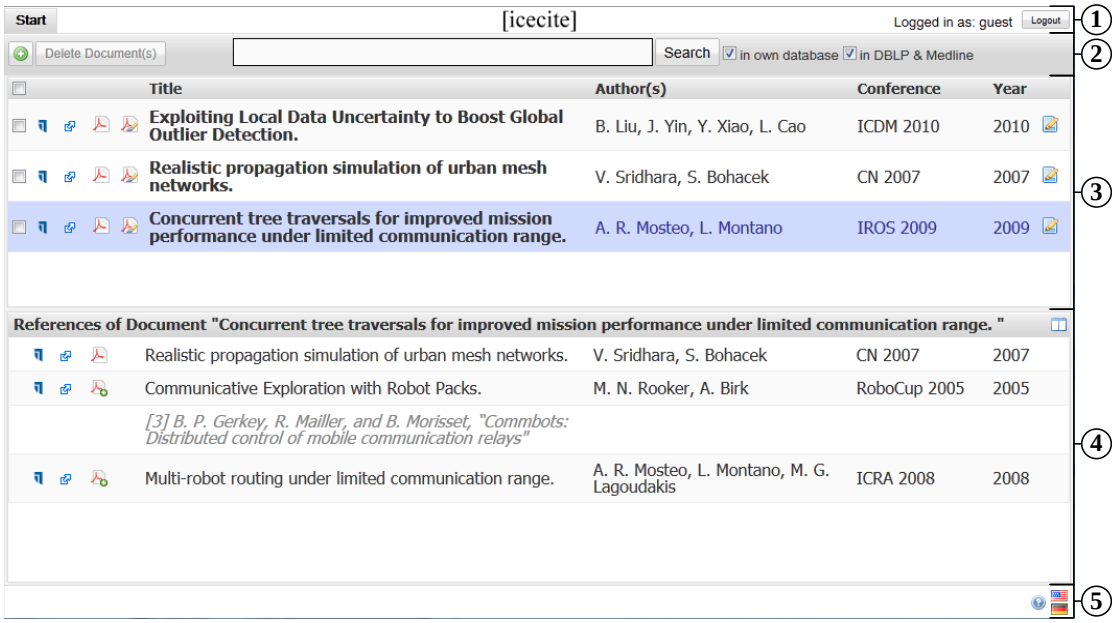


Abbildung 6.12: Die Bibliothekssicht der Anwendung unterteilt sich in die Kopfzeile (Bereich 1), in die Werkzeugleiste (Bereich 2), in die Publikationsliste (Bereich 3), in die Referenzenliste (Bereich 4) und in die Fußzeile (Bereich 5).

Kopfzeile

In der Kopfzeile ist neben dem zentral platzierten Logo der Anwendung („icecite“ ist der Arbeitstitel der Anwendung) der Name des eingeloggten Benutzers und ein Logout-Button zum Beenden der Sitzung zu finden. Die Kopfzeile ist ein statisches Element der Anwendung, d.h. sie wird immer angezeigt – egal auf welcher Seite sich der Benutzer befindet.

Werkzeugleiste

Mit Klick auf kann der Benutzer Publikationen zu seiner Bibliothek hinzufügen, mit Klick auf Delete Document(s) Publikationen aus seiner Bibliothek löschen und durch Eingabe einer Suchanfrage in das Textfeld sowohl DBLP und Medline als auch seine eigene Bibliothek durchsuchen.

Hinzufügen von Publikationen Um Publikationen seiner Bibliothek hinzuzufügen, kann der Benutzer die auf seinem Rechner gespeicherte PDF-Dateien wissenschaftlicher Publikationen hochladen, aus denen das System sowohl den Titel als auch die Referenzen extrahiert und Einträgen aus DBLP oder Medline zuordnet. Im Folgenden sei der gesamte Vorgang beispielhaft an den beiden PDF-Dateien `10.1.1.265.pdf` und `10.1.1.590.pdf` beschrieben (vgl. Abbildung 6.13).

Abbildung 6.13: Anhand der Dateinamen `10.1.1.265.pdf` und `10.1.1.590.pdf` lassen sich die Publikationen nur schwer identifizieren. Statt die PDF-Dateien in einem lokalen Dateisystem zu verwalten, können sie in eine persönliche Bibliothek unserer Anwendung geladen werden, in der sie für eine eindeutige Identifizierung mit vollständigen Metadaten beschrieben werden.

Angenommen, der Benutzer wisse zwar, dass es sich bei beiden Dateien um wissenschaftliche Publikationen handelt, er aber anhand der Dateinamen nicht feststellen könne, um welche Publikationen es sich konkret handelt. Der Benutzer lädt beide PDF-Dateien hoch, indem er auf klickt und beide PDF-Dateien im sich öffnenden Dateidialog auswählt. Das System speichert die Dateien auf dem Server zunächst in `~/uploads/<user>/tmp`. Um mögliche Namenskonflikte zu vermeiden, wird jede PDF-Datei umbenannt und mit einem Zeitstempel versehen (vgl. Beispiel 6.14). Für jede hochgeladene Datei wird nun der Titel, wie in Kapitel 3.2 bespro-

ursprünglicher Dateiname:	`10.1.1.265.pdf`
Muster temporärer Dateiname:	`<user>-<Zeitstempel>.pdf`
temporärer Dateiname:	`korzen-560305018.pdf`

Beispiel 6.14: Das temporäre Dateinamen-Muster für PDF-Dateien. Um mögliche Namenskonflikte zu vermeiden, erhält der Dateiname neben dem Namen des Benutzers zusätzlich einen Zeitstempel.

chen, extrahiert. Mit diesem Titel stellt das System eine Anfrage an den Z-Server, der mit einer definierten Anzahl Keys von Literaturdatenbank-Einträgen antwortet. Anschließend wird mit jedem erhaltenen Key eine Anfrage an den DS-Server gestellt, um jeweils die vollständigen Metadaten zu dem entsprechenden Eintrag zu erhalten.

Genauso wird auch mit den Referenzen verfahren: Zunächst werden alle Referenzen, wie in Kapitel 3.3 besprochen, extrahiert und mit jeder extrahierten Referenz eine Anfrage an den Z-Server gestellt. Mit jedem erhaltenen Key stellt das System eine Anfrage an den DS-Server um jeweils die vollständigen Metadaten zu erhalten. Gleichzeitig erstellt das System von der PDF-Datei eine *annotierte Version*, in der alle identifizierten Referenzen in der PDF-Datei eine Annotation erhalten. Jede Annotation kann mit einer beliebigen, individuellen URL versehen werden – wie z.B. mit einer URL zu einer Google-Suche mit den Autoren und dem Titel der Referenz als Suchanfrage (vgl. Abbildung 6.15). Während der Benutzer das Literaturverzeichnis einer annotierten Publikation liest, kann er also per Klick auf eine Annotation eine entsprechende Google-Suche starten, um z.B. weitere Informationen zur Referenz zu erhalten.

Schließlich wird mithilfe des `convert`-Werkzeuges der ImageMagick-Bibliothek [14] für jede hochgeladene PDF-Datei eine jpg-Grafik von der ersten Seite erstellt. Mit dem Befehl

```
convert -resize x750 <PDF-Dateiname>[0] ~/uploads/<user>/tmp
```

erstellt das System eine jpg-Grafik mit der Höhe von 750 Pixel (`x750` definiert eine feste Höhe von 750 Pixel, wobei die Breite proportional angepasst wird) von der ersten Seite ([0] spezifiziert die erste Seite der PDF-Datei) der definierten PDF-Datei und speichert sie im Ordner `~/uploads/<user>/tmp` ab.

Da wir uns prinzipiell nicht darauf verlassen können, stets den korrekten Eintrag einer Publikation in den Literaturdatenbanken zu finden, wird dem Benutzer nun in einem Fenster für jede hochgeladene PDF-Datei eine Liste von fünf möglichen Zuordnungs-Kandidaten für

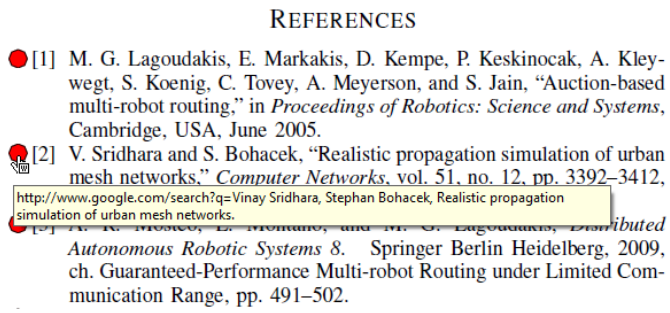


Abbildung 6.15: Die Annotationen von Referenzen in einer annotierten Publikation. Jede Annotation enthält eine individuelle URL zu einer Google-Suche mit dem Titel und den Autoren der Referenz als Suchanfrage.

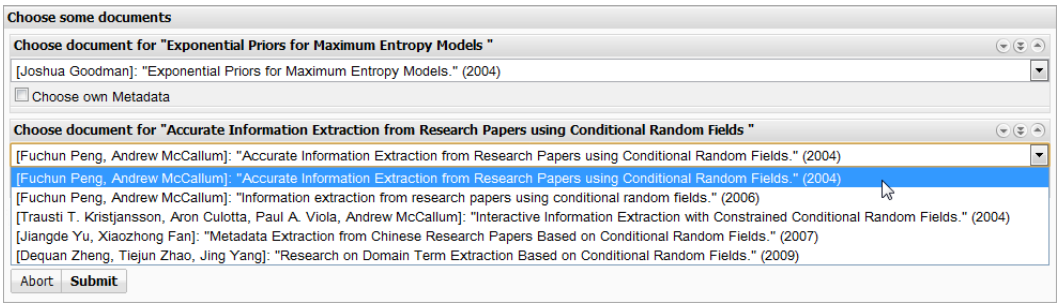


Abbildung 6.16: Fenster, in dem für jede hochgeladene PDF-Datei der korrekte Literaturdatenbank-Kandidat ausgewählt kann. Jeder Kandidat wird mit seinen Autoren, seinem Titel und seinem Erscheinungsjahr in einer *Dropdown-Liste* aufgelistet.

den Titel angezeigt (vgl. Abbildung 6.16). Aus den bereitgestellten Listen kann der Benutzer den korrekten Kandidaten auswählen. Um tatsächlich eine korrekte Entscheidung treffen zu können, ist es dem Benutzer zudem möglich, die erstellte jpg-Grafik der ersten Seite einer PDF-Datei einzublenden. Mit Klick auf kann sich der Benutzer entweder das obere Drittel oder mit Klick auf die gesamte Grafik anzeigen lassen. Aus ihr kann er den Titel und die Autoren ablesen und damit seine Entscheidung überprüfen (vgl. Abbildung 6.17).

Für den Fall, dass entweder kein vorgeschlagener Kandidat korrekt ist oder kein Eintrag in beiden Literaturdatenbanken gefunden wurde, hat der Benutzer zudem die Möglichkeit, durch Wählen der Option „Choose own Metadata" eigene Metadaten in einem sich öffnenden Formular anzugeben. Nach Bestätigung der Angaben wird jede PDF-Datei entsprechend des Keys des ausgewählten Zuordnungs-Kandidaten nach dem Muster aus Beispiel 6.18 umbenannt und in `~/uploads/<user>/` verschoben. Alle temporären Dateien werden gelöscht (z.B. die Grafiken der ersten Seiten) und entsprechende Fragment-Dateien in `~/uploads/<user>/xml` erstellt. Daraufhin setzt das System `library.xml` neu zusammen und liest diese ein, um den Index für den BS-Server neu zu bauen und um die Metadaten in der Bibliothekssicht anzuzeigen.

Key des ausgewählten Kandidaten:	`conf/naacl/PengM04`
Dateinamen für PDF-Dateien:	`conf.naacl.PengM04.pdf`
	`conf.naacl.PengM04_annot.pdf`

Beispiel 6.18: Das Muster für die endgültigen Dateinamen der PDF-Dateien. Der Dateiname resultiert aus dem Key des ausgewählten Kandidaten, indem das Zeichen `"/"` durch das Zeichen `"."` ersetzt und die Dateiendung `".pdf"` angehängt wird. Die annotierten Publikationen erhalten vor der Dateiendung den Zusatz `"_annot"`.

Der Vorgang des Hinzufügens neuer Publikationen ist damit abgeschlossen. Zu beachten ist, dass wir uns dazu entschlossen haben, den Benutzer zwar den richtigen Kandidaten für den Titel auswählen zu lassen, jedoch nicht die richtigen Kandidaten für die Referenzen. Während wir durch die Auswahl eines Kandidaten für den Titel durch den Benutzer sicher gehen wollen, dass tatsächlich die korrekten Metadaten für die Publikation in der Bibliothek hin-

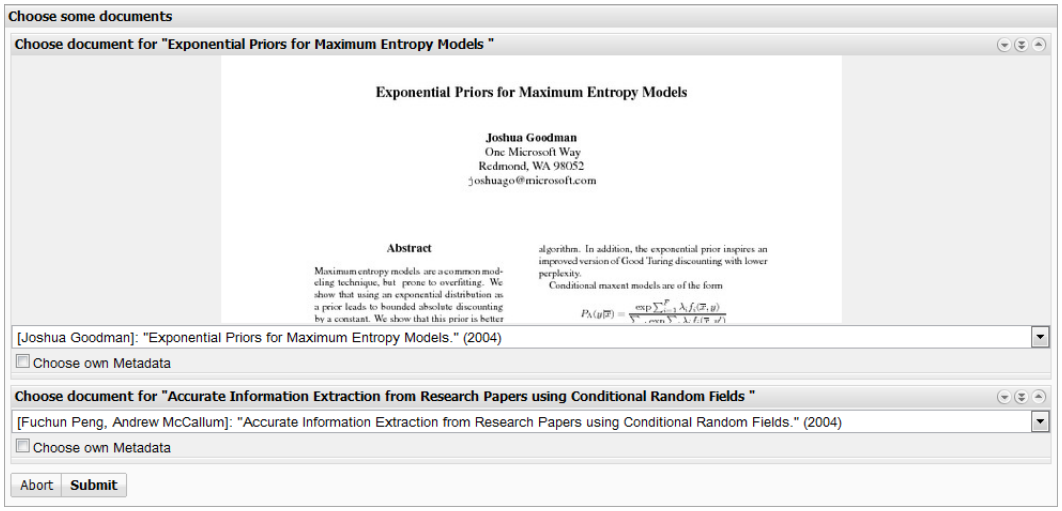


Abbildung 6.17: Für die Auswahl des korrekten Kandidaten kann die erste Seite (entweder das obere Drittel oder die komplette Seite) der Publikation eingeblendet werden, aus der der tatsächliche Titel und die Autoren abgelesen werden können. Eigene Metadaten können nach dem Aktivieren der Checkbox „Choose Own Metadata“ in einem sich öffnenden Formular eingegeben werden.

terlegt werden, empfanden wir den Aufwand für den Benutzer, auch für jede Referenz den korrekten Kandidaten identifizieren zu müssen, als zu hoch. Stattdessen wählen wir für eine Referenz denjenigen Zuordnungs-Kandidaten aus, für den die Wahrscheinlichkeit, der korrekte Literaturdatenbank-Eintrag zu sein, am höchsten ist – also den ersten Kandidaten aus der Kandidatenliste. Wir nehmen dadurch allerdings in Kauf, dass einige Referenzen möglicherweise falsch zugeordnet werden.

Löschen von Publikationen Durch Klick auf Delete Document(s) kann der Benutzer alle durch ein Häkchen ausgewählte Publikationen aus der Bibliothek entfernen. Dabei werden sowohl die entsprechenden PDF-Dateien in `~/uploads/<user>/` als auch die entsprechenden Fragment-Dateien in `~/uploads/<user>/xml` gelöscht. Die XML-Datei `library.xml` wird aus den verbliebenen Fragment-Dateien neu zusammengesetzt und daraufhin neu ausgelesen.

Suche Durch die Eingabe einer Suchanfrage in das Textfeld der Werkzeugleiste kann der Benutzer je nach Auswahl der Punkte „in own database“ und „in DBLP & Medline“ seine eigene Bibliothek und/oder die Literaturdatenbanken DBLP und Medline durchsuchen. Während die Suche in den Literaturdatenbanken vom DS-Server übernommen wird, ist für die Suche in der Bibliothek eines Benutzers ein individueller BS-Server verantwortlich. Der dem BS-Server zugrunde liegende Index wird über die Inhalte aus `~/uploads/<user>/library.xml` erstellt. Jedes Mal, wenn ein Benutzer sich im System anmeldet, wird ein BS-Server mit diesem Index an einem freien Port gestartet.

Wenn sich die Inhalte in `library.xml` ändern (also wenn der Benutzer Publikationen hinzufügt, löscht oder Metadaten editiert), wird der Index für den BS-Server neu erstellt. Dieser Prozess läuft nebenläufig im Hintergrund ab und beeinflusst die aktuell laufende Instanz des BS-Server zunächst nicht. Sobald der Index-Neubau abgeschlossen ist, wird ein Prozess für einen Neustart des BS-Servers erstellt. Erst kurz bevor die neue Instanz des BS-Servers am gleichen Port gebunden wird, wird die aktuelle Instanz anhand seiner Prozess-ID (PID) beendet und die neue Instanz gestartet. Die Zeit, in der der BS-Server durch den Neustart für den Benutzer nicht zur Verfügung steht, wird dadurch minimal gehalten. Von nun an wird für Suchanfragen des Benutzers die neue Instanz des BS-Servers verwendet. Beendet der User schließlich seine Sitzung, so wird seine Instanz des BS-Servers anhand der PID identifiziert und beendet, so dass der verwendete Port wieder freigegeben wird. Ebenso wie der DS-Server antwortet der BS-Server auf Suchanfragen mit einem XML-Dokument, das alle Einträge der Bibliothek

enthält, die der Suchanfrage entsprechen.

Wenn für eine Suche sowohl der DS-Server als auch der BS-Server bemüht wurden, werden beide Suchergebnisse zusammengefügt und in der Publikationsliste der Bibliothekssicht angezeigt. Dabei werden die Suchergebnisse des BS-Servers stets vor den Suchergebnissen des DS-Servers aufgelistet. Da mit dem BS-Server zusätzlich die Volltexte von Publikationen durchsucht werden, werden seine Suchergebnisse zusammen mit sogenannten *Excerpts* angezeigt. Ein Excerpt ist ein kleiner Textauszug des Volltextes, der relevant zur Suchanfrage ist. So lässt sich auf einen Blick nachvollziehen, warum sich ein bestimmter Eintrag unter den Suchergebnissen befindet.

Publikationsliste

In der Publikationsliste werden entweder alle Publikationen der Bibliothek oder im Falle einer Suche die entsprechenden Suchergebnisse angezeigt. Neben ihren Metadaten wie Titel, Autoren, Konferenz und Erscheinungsjahr enthält jeder Listeneintrag eine Reihe von Symbolen mit jeweils unterschiedlichen Bedeutungen:

Das Symbol verweist auf diejenige URL, die für einen Eintrag in DBLP im Feld `ee` hinterlegt ist. Nach einem Klick auf dieses Symbol gelangt der Benutzer auf die Webseite des Verlegers der Publikation. Hier wird neben weiteren Informationen zur Publikation meist eine PDF-Version der Publikation vertrieben. Dagegen verweist das Symbol auf die URL, die in DBLP im Feld `url` hinterlegt ist. Bei Klick auf dieses Symbol erreicht der Benutzer die DBLP-interne Seite der Publikation. Auch hier gibt es zusätzliche Informationen zur Publikation, wie z.B. einen BibTeX-Eintrag, bestückt mit den Metadaten der Publikation. Bei Klick auf eines der PDF-Symbole gelangt der Benutzer zur *Detailsicht* (s. Kapitel 6.4.3) einer Publikation. Hier wird ihm entweder die normale PDF-Version (bei Klick auf) oder die annotierte PDF-Version (bei Klick auf) angezeigt.

Durch den Klick auf den Titel einer Publikation werden dem Benutzer die entsprechenden Referenzen in der Referenzenliste angezeigt. Er kann zudem alle Metadaten der Publikationen durch Klick auf das Symbol editieren. Ähnlich wie beim Hinzufügen von Publikationen kann der Benutzer dazu einen (alternativen) Kandidaten aus einer Liste auswählen oder die bereits hinterlegten Metadaten manuell ändern. Nach Bestätigung der Änderungen identifiziert das System anhand des Keys der Publikation die zu bearbeitende Fragment-Datei und schreibt diese komplett neu. Falls der Benutzer einen alternativen Kandidaten ausgewählt hat, ändert sich dementsprechend auch der zugeordnete Key. Da wir sowohl die PDF-Dateien als auch die Fragment-Dateien jeweils entsprechend des Keys benannt haben, müssen wir diese Dateien entsprechend dem neuen Key umbenennen.

Referenzenliste

In der Referenzenliste werde alle Referenzen einer ausgewählten Publikation angezeigt. Wenn eine Referenz nicht zugeordnet werden konnte, stehen uns für diese keine Metadaten zur Verfügung. Der entsprechende Eintrag in der Referenzenliste enthält deshalb lediglich die extrahierte Zeichenkette in grauer Schriftfarbe. Ansonsten ist der Aufbau ähnlich dem Aufbau der Publikationsliste. Die Symbole und haben die gleiche Bedeutung wie in der Publikationsliste. Das Symbol wird für eine Referenz nur dann angezeigt, wenn sie bereits als Publikation in der Bibliothek gespeichert ist. Dies stellen wir fest, indem wir überprüfen, ob zu dem Key der Referenz bereits eine Fragment-Datei in `~/uploads/<user>/xml` existiert. Ist dies der Fall, existiert die Referenz bereits als Publikation in der Bibliothek und der Benutzer gelangt nach Klick auf dieses Symbol zur Detailsicht der Publikation, wo ihm die entsprechende PDF-Datei angezeigt wird.

Falls die Referenz noch nicht als Publikation in der Bibliothek hinterlegt ist, so wird das Symbol eingeblendet. Bei Klick auf dieses Symbol versucht das System, eine entsprechende PDF-Datei für diese Referenz auf folgende Weise im Internet zu finden und der Bibliothek hinzuzufügen:

Automatisches Hinzufügen von Publikationen Um die PDF-Datei einer Publikation im Internet zu finden, verwenden wir die URL, die im Feld `ee` von DBLP (vgl. Kapitel 4.1) gespeichert ist. Zum Teil verweist diese `ee`-URL direkt auf die PDF-Datei der Publikation. Weil die meisten Publikationen aber aus Copyright-Gründen vor unbefugtem Zugriff geschützt werden, verweist sie meistens auf eine Seite des Verlegers, auf der man über eine von der `ee`-URL verschiedenen Download-URL die PDF-Datei herunterladen kann, falls man die notwendigen Zugriffsrechte hat. Je nach Verleger kann das System die Download-URL der PDF-Datei aus der `ee`-URL herleiten, so dass es die PDF-Datei herunterladen kann und der Bibliothek hinzufügen kann. Dies soll an einigen Beispielen aufgezeigt werden[1]:

- **Springer:** Ist der Verleger einer Publikation Springer [22], so lautet die `ee`-URL beispielsweise:

  `http://www.springerlink.com/content/c873271685124v42/`

  Die entsprechende Download-URL für die PDF-Datei lautet dann

  `http://www.springerlink.com/content/c873271685124v42/fulltext.pdf`

  Wir müssen also lediglich `"fulltext.pdf"` an die `ee`-URL anhängen, um die Download-URL der PDF-Datei zu erhalten.

- **ACM:** Ist der Verleger einer Publikation ACM [1], so enthält das Feld `ee` eine URL beispielsweise der Form

  `http://dl.acm.org/citation.cfm?doid=1883612.1883613`

  Die entsprechende Download-URL für die PDF-Datei lautet dann

  `http://dl.acm.org/ft_gateway.cfm?id=1883613`

  Um diese zu erhalten, entnehmen wir der `ee`-URL den Wert des Parameters `doid`. Dieser Wert enthält zwei ID's, die durch einen Punkt getrennt sind. Wir entnehmen diesem Wert die zweite ID, die wir als Parameter `id` in der Download-URL verwenden.

- **IEEE:** Ist der Verleger einer Publikation IEEE [13], so lautet die `ee`-URL beispielsweise

  `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5767718`

  Die entsprechende Download-URL für die PDF-Datei lautet dann

  `http://ieeexplore.ieee.org/ielx5/2/5767713/05767718.pdf?arnumber=5767718&isnumber=5767713`

  Die Herleitung dieser Download-URL ist deutlich komplizierter, als die vorherigen Fälle, da sie mehr notwendige Parameter enthält, als wir der `ee`-URL entnehmen können. Dennoch können wir sie uns durch folgende alternative Herangehensweise herleiten: Der Quellcode der Webseite, zu der die `ee`-URL verweist, enthält im Header-Bereich ein `meta`-Element[2] mit dem Namen `citation_pdf_url`, das als Inhalt eine URL

  `http://ieeexplore.ieee.org/iel5/2/5767713/05767718.pdf?arnumber=5767718`

  enthält. Diese ist der Download-URL schon recht ähnlich. Aus ihr können wir den Wert für den uns bisher unbekannten Parameter `isnumber` (`5767713`) entnehmen. Um schließlich die korrekte Download-URL zu erhalten, ändern wir die Teilzeichenkette `iel5` der `meta`-URL durch Hinzufügen des Zeichens `"x"` zu `ielx5` und hängen der `meta`-URL den Parameter `isnumber` zusammen mit dem eben bestimmten Wert (also die Zeichenkette `"&isnumber=5767713"`) an.

  Alleine die Anfrage mit dieser Download-URL reicht aber noch nicht aus, um die PDF-Datei herunterladen zu können. Zusätzlich brauchen wir ein Cookie mit dem Namen `ERIGHTS`, den wir am Anfang der Sitzung von IEEE zugewiesen bekommen. Zusammen mit diesem Cookie können wir dann die gewünschte PDF-Datei herunterladen.

Haben wir die Download-URL der PDF-Datei ermittelt, fordern wir die Datei mit einer GET-Anfrage (vgl. Beispiel 6.19) vom jeweiligen Server an. Wenn die Download-URL korrekt

```
GET /content/c873271685124v42/fulltext.pdf HTTP/1.1
Host: www.springerlink.com
User-Agent: Mozilla/5.0
```

Beispiel 6.19: GET-Anfrage, um die PDF-Datei `/content/c873271685124v42/fulltext.pdf` vom Host `www.springerlink.com` anzufordern.

ist, erhalten wir vom Server des Verlegers als Antwort eine PDF-Datei, die wir zunächst in `~/uploads/<user>/tmp` zwischenspeichern.

Um sie der Bibliothek hinzuzufügen, müssen nun lediglich die Referenzen extrahiert und zugeordnet werden, da die Metadaten der Publikation bereits bekannt sind (nämlich weil es sich bei der Publikation um eine Referenz einer bereits hinzugefügten Publikation handelt und die Metadaten dieser Referenz bereits beim Hinzufügen der Publikation bestimmt wurden). Damit fällt auch die notwendige Auswahl des korrekten Kandidaten durch den Benutzer weg. Der Benutzer erhält dadurch eine weitere, sehr einfache und sehr komfortable Möglichkeit, Publikationen seiner Bibliothek hinzuzufügen. Im Idealfall muss er nur ein einziges Mal (nämlich dann, wenn noch keine Publikation in seiner Bibliothek hinterlegt ist) eine PDF-Datei hochladen und einen Kandidaten auswählen. Alle weiteren Publikationen können dank der gerade beschriebenen Funktionalität mit nur einem Klick hinzugefügt werden.

Zu beachten ist, dass der automatische Download von Publikationen zurzeit lediglich für DBLP-Publikationen möglich ist, weil Medline keine vergleichbaren URL's zu Publikationen speichert. Aber auch für DBLP-Publikationen gelingt der Download nicht immer. Prinzipiell gelingt er nur dann, wenn für eine gegebene `ee`-URL bekannt ist, wie die entsprechende Download-URL herzuleiten ist. Aber auch wenn die Download-URL aus der `ee`-URL abzuleiten ist, ist es möglich, dass der Download von PDF-Dateien fehlschlägt. Wenn zum Beispiel die Struktur der Download-URL von den üblichen Konventionen des Verlegers abweicht oder generell keine PDF-Datei für die Publikation verfügbar ist, kann keine PDF-Datei heruntergeladen werden. Es kann also nicht garantiert werden, dass das automatische Hinzufügen von Publikationen stets gelingt.

Fußzeile

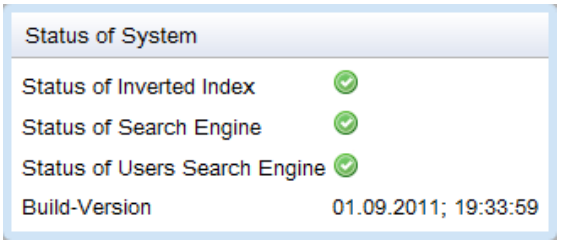


Abbildung 6.20: Fenster mit Informationen über den Status des Systems, aus dem die Erreichbarkeit des Z-Servers („Inverted Index"), des DS-Servers („Search Engine") und des BS-Servers („User's Search Engine") sowie das Datum der letzten Aktualisierung der Anwendung ablesbar ist.

Ebenso wie die Kopfzeile ist die Fußzeile ein statisches Element der Anwendung. Sie dient in erster Linie dazu, dem Benutzer Nachrichten über bestimmte Ereignisse (z.B. Hinweise oder den Fortschritt des Uploads von PDF-Dateien) einzublenden. Bei Klick auf öffnet sich ein Fenster, in dem Informationen zum Status des Systems zu finden sind (vgl. Abbildung 6.20).

[1]Da sich der Server unserer Anwendung im Netzwerk der Technischen Fakultät der Universität Freiburg befindet, haben wir meistens die notwendigen Zugriffsrechte auf die PDF-Dateien. Wenn es uns gelingt, die jeweilige Download-URL zu bestimmen, hat somit theoretisch auch jeder Benutzer unserer Anwendung Zugriff auf diese Dateien. Da die Anmeldung in unsere Anwendung an ein Benutzerkonto der Technischen Fakultät gebunden ist, sind alle Benutzer zugleich Mitglied der Technischen Fakultät und besitzen damit auch die notwendigen Zugriffsrechte. Wir umgehen die Zugriffskontrollen der Verleger deshalb nicht.

[2]HTML-Element, mit dem die Metadaten einer HTML-Seite angegeben werden können:
`<meta name="citation_pdf_url"content="http://ieeexplore.ieee.org/iel5/2/5767713/[...]"/>`

Hier kann sich der Benutzer informieren, ob der Z-Server (in der Abbildung „Inverted Index“ genannt), der DS-Server („Search Engine“) und der BS-Server („User's Search Engine“) erreichbar sind. Zudem ist das Datum der letzten Aktualisierung der Anwendung ablesbar. Über die Symbole und kann der Benutzer zudem die Sprache des Systems auf Deutsch bzw. Englisch umstellen.

6.4.3 Die Detailsicht

Sobald der Benutzer auf eines der PDF-Symbole einer Publikation oder einer Referenz klickt, wird er auf die *Detailsicht* der Publikation geleitet (Abbildung 6.21). Falls für den Browser des

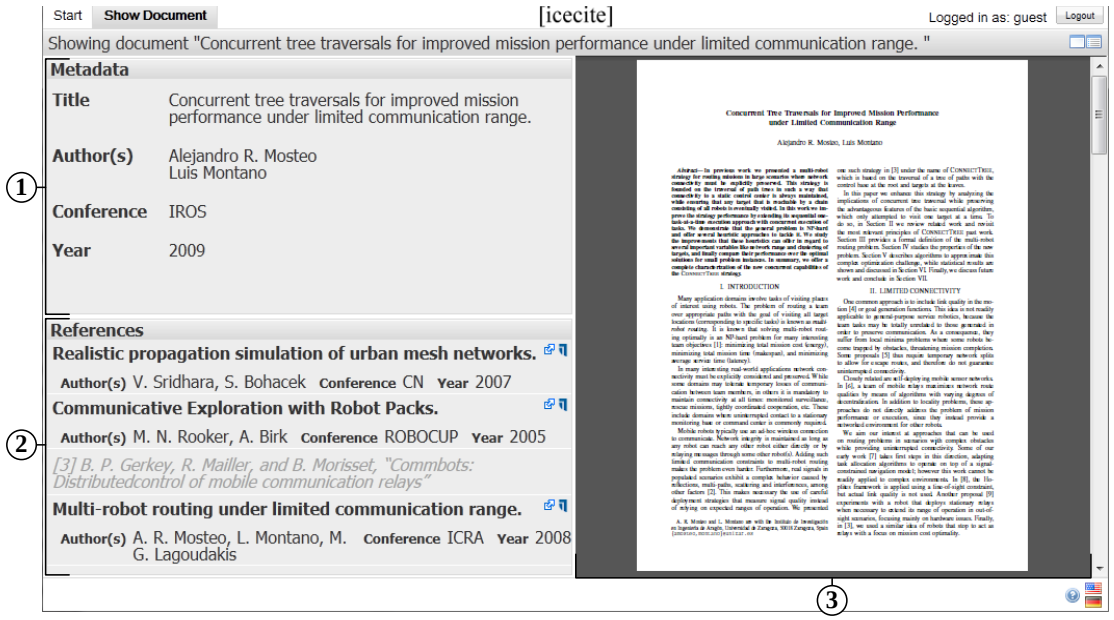


Abbildung 6.21: Die Detailsicht der Anwendung teilt sich in die Metadatenliste (Bereich 1), in die Referenzenliste (Bereich 2) und in die PDF-Anzeige (Bereich 3) auf.

Benutzers ein entsprechendes PDF-Plugin installiert ist, wird ihm hier entweder die normale Version (bei Klick auf) oder die annotierte Version (bei Klick auf) der PDF-Datei angezeigt (vgl. Bereich 3 in Abbildung 6.21) – zusammen mit den Metadaten (Bereich 1) und den Referenzen (Bereich 2). Durch Klick auf kann der Benutzer die Metadaten ausblenden, so dass die Anzeige der PDF-Datei die gesamte Breite des Bildschirms füllt, oder durch Klick auf die Anzeige der PDF-Datei ausblenden, so dass nur die Metadaten und die Referenzen angezeigt werden.

Ist kein geeignetes PDF-Plugin verfügbar, so kann der Benutzer die PDF-Datei alternativ herunterladen, um sie in einem externen PDF-Betrachter öffnen zu können.

Kapitel 7

Zusammenfassung

Wir haben in dieser Masterarbeit ein System vorgestellt, mit der alle notwendigen Schritte einer Literaturrecherche und Literaturverwaltung wesentlich vereinfacht und beschleunigt werden können. Eine entscheidende Voraussetzung für das System war die Extraktion von Titeln und Referenzen aus PDF-Dateien wissenschaftlicher Publikationen. Um vollständige und korrekte Metadaten der Publikationen und ihrer Referenzen zu erhalten, haben wir zusätzlich die extrahierten Zeichenketten Einträgen aus den Literaturdatenbanken DBLP und Medline zugeordnet. Die resultierenden Metadaten haben wir dazu genutzt, die Publikationen einer persönlichen Bibliothek eindeutig zu beschreiben und intuitiv identifizierbar zu machen. Zudem konnten sowohl die Metadaten und die Volltexte der Publikationen als auch *DBLP* und *Medline* durchsucht werden, so dass eine gezielte Suche nach bestimmten Publikationen möglich war. Annotierte Publikationen wurden dazu erstellt, um auch während dem Lesen eines Literaturverzeichnisses durch Klick auf eine Annotation weitere Informationen zu referenzierten Publikationen zu erhalten. Jede Bibliothek konnte komfortabel erweitert werden, indem per Knopfdruck automatisch eine PDF-Datei einer ausgewählten Referenz im Internet gesucht und der Bibliothek hinzugefügt wurde.

Da für uns präzise Daten und kurze Antwortzeiten der Web-Anwendung von entscheidender Bedeutung waren, haben wir alle Algorithmen und Komponenten unter den Aspekten der Korrektheit und der Effizienz implementiert. Die Ergebnisse ausführlicher Experimente zur Evaluation unserer Algorithmen haben aber gezeigt, dass beide Aspekte nicht immer miteinander vereinbar sind: Besonders bei der Titel- und Referenzen-Zuordnung konnten wir eine höhere Korrektheit der Zuordnungs-Ergebnisse nur mit einer gleichzeitigen Erhöhung der Laufzeiten erreichen: Im schlechtesten Fall bedeutete z.B. eine Erhöhung der Korrektheit der Ergebnisse der Titel-Zuordnung von DBLP-Publikationen um 1,7% (von 98,1% auf 99,9%) gleichzeitig eine Erhöhung der Laufzeit um über 330% (von 4,93ms auf 21,45ms). Im günstigsten Fall stieg die Laufzeit bei der Referenzen-Zuordnung für Medline-Publikationen z.B. lediglich um 40,73% (von 26,96ms auf 37,94ms), wobei die Korrektheit der Ergebnisse um 13,6% (von 83,1% auf 93,6%) verbessert werden konnte.

Die Experimente zur Evaluation der Titel- und Referenzen-Extraktion ergaben schließlich, dass wir den Titel für 98,8% aller DBLP- und Medline-Publikationen in durchschnittlich 43ms extrahieren konnten. Die Extraktion der Referenzen gelang uns für 81,6% der DBLP-Publikationen in 141,85ms und für 91,1% der Medline-Publikationen in 170,75ms.

Ausblick

Das in dieser Masterarbeit implementierte System stellt eine geeignete Grundlage zu einer einfachen und komfortablen Literaturrecherche dar. Zudem sind dank des modularen Aufbau des Systems und die Verwendung des Entwurfsmusters *Model-View-Presenter* bei der Implementierung unserer Web-Anwendung alle Komponenten beliebig erweiterbar.

Verbesserungsbedarf sehen wir vor allem bei der Geschwindigkeit der Text-Extraktion aus PDF-Dateien, da die Aufgaben der zurzeit verwendeten Java-Bibliothek PDFBox mit über 70%

den größten Anteil der Laufzeiten für die Titel- und Referenzen-Extraktion ausmachen. Aufgrund der äußerst komplexen PDF-Spezifikationen ist dieses Vorhaben aber keineswegs ein triviales.

Zudem ist der Algorithmus der Referenzen-Extraktion so zu optimieren, dass auch Referenzen nach Einschüben in Literaturverzeichnissen (vgl. Beispiel B.5) identifiziert werden können. Neben möglichen Optimierungen für den bestehenden Code sehen wir zudem einige Möglichkeiten, die Anwendung mit zusätzlichen Funktionalitäten zu erweitern.

So könnte zum Beispiel mit einem zu entwickelndem Plugin für Webbrowser eine Publikation, die gerade innerhalb des Browsers geöffnet ist, seiner Bibliothek per Knopfdruck hinzufügen. Denkbar wäre auch, die Annotationen der Referenzen innerhalb der annotierten Version einer Publikation so zu modifizieren, dass per Klick auf eine Annotation die referenzierte Publikation zur Bibliothek hinzugefügt wird. Zusätzlich könnte das automatische Herunterladen und Hinzufügen von Publikationen zur Bibliothek auf Medline-Publikationen ausgeweitet werden. Zwar speichert Medline keine zur `ee`-URL vergleichbare URL, aber für jede Publikation existiert in *PubMed* eine individuelle Seite, die unter der URL

`http://www.ncbi.nlm.nih.gov/pubmed/<Medline-Key>`

zu finden ist. Für einige Publikationen sind hier URL's zu entsprechenden PDF-Dateien zu finden. Durch Aufruf solch einer PubMed-Seite kann also überprüft werden, ob solch eine Download-URL existiert und ob gegebenenfalls eine PDF-Datei der Publikation heruntergeladen werden kann, um sie der Bibliothek hinzufügen zu können.

Alle Optimierungen und Erweiterungen der Anwendung können schließlich dazu beitragen, eine Literaturrecherche noch komfortabler zu gestalten.

Literaturverzeichnis

[1] ACM Digital Library. http://dl.acm.org/,

[2] Apache PDFBox - Java PDF Library. http://pdfbox.apache.org/,

[3] Citavi: Reference Manager and Knowledge Organization. http://www.citavi.com/,

[4] CiteSeer$^{\chi}$. http://citeseerx.ist.psu.edu/,

[5] CiteULike: Everyone's library. http://www.citeulike.org/,

[6] CompleteSearch DBLP. http://www.dblp.org/search/,

[7] DBLP FAQ: What is the meaning of "DBLP"? http://www.informatik.uni-trier.de/~ley/db/about/faqdblp.html,

[8] dblp.xml. http://dblp.uni-trier.de/xml/,

[9] EndNote - Bibliographies Made Easy. http://www.endnote.com/,

[10] Google Scholar. http://scholar.google.de/,

[11] Google Web Toolkit. http://code.google.com/intl/de-DE/webtoolkit/,

[12] HttpUnit. http://httpunit.sourceforge.net/,

[13] IEEE Xplore. http://ieeexplore.ieee.org/Xplore/,

[14] ImageMagick: Convert, Edit, Or Compose Bitmap Images. http://www.imagemagick.org/script/index.php,

[15] JabRef reference manager. http://jabref.sourceforge.net/,

[16] Mendeley Feedback Forum. Thema: „Version 0.9.7 does not extract references from the pdf file". http://feedback.mendeley.com/forums/4941-mendeley-feedback/suggestions/834313-version-0-9-7-does-not-extract-references-from-the,

[17] Mendeley: Free reference manager and PDF organizer. http://www.mendeley.com/,

[18] PubMed - NCBI. http://www.ncbi.nlm.nih.gov/pubmed/,

[19] Rexa. http://rexa.info/,

[20] SAX Project. http://www.saxproject.org/,

[21] Schriftauszeichnung - Wikipedia. http://de.wikipedia.org/wiki/Schriftauszeichnung,

[22] SpringerLink. http://www.springerlink.com/,

[23] U.S. National Library of Medicine: Data, News and Update Information. http://www.nlm.nih.gov/bsd/revup/revup_pub.html#med_update,

[24] W3C Document Object Model. http://www.w3.org/DOM/,

[25] Zotero. http://www.zotero.org/,

[26] BAST, Hannah ; WEBER, Ingmar: Type less, find more: fast autocompletion search with a succinct index. In: SIGIR, 2006, S. 364–371

[27] BAST, Hannah ; WEBER, Ingmar: The CompleteSearch Engine: Interactive, Efficient, and Towards IR&DB Integration. In: CIDR, 2007, S. 88–95

[28] BEEL, Jöran ; GIPP, Bela ; SHAKER, Ammar ; FRIEDRICH, Nick: SciPlore Xtract: Extracting Titles from Scientific PDF Documents by Analyzing Style Information (Font Size). In: ECDL, 2010, S. 413–416

[29] BOLLACKER, Kurt D. ; LAWRENCE, Steve ; GILES, C. L.: CiteSeer: An Autonous Web Agent for Automatic Retrieval and Identification of Interesting Publications. In: Agents, 1998, S. 116–123

[30] BORKAR, Vinayak R. ; DESHMUKH, Kaustubh ; SARAWAGI, Sunita: Automatic Segmentation of Text into Structured Records. In: SIGMOD Conference, 2001, S. 175–186

[31] CONNAN, J. ; OMLIN, C. W.: Bibliography Extraction with Hidden Markov Models. 2000. – Forschungsbericht

[32] GILES, C. L. ; BOLLACKER, Kurt D. ; LAWRENCE, Steve: CiteSeer: An Automatic Citation Indexing System. In: ACM DL, 1998, S. 89–98

[33] GUSFIELD, Dan: Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press, 1997. – ISBN 0–521–58519–8

[34] HAN, Hui ; GILES, C. L. ; MANAVOGLU, Eren ; ZHA, Hongyuan ; ZHANG, Zhenyue ; FOX, Edward A.: Automatic Document Metadata Extraction Using Support Vector Machines. In: JCDL, 2003, S. 37–48

[35] HENNING, Victor ; REICHELT, Jan: Mendeley - A Last.fm For Research? In: eScience, 2008, S. 327–328

[36] HETZNER, Erik: A simple method for citation metadata extraction using hidden markov models. In: JCDL, 2008, S. 280–284

[37] KRATZER, Mathias: Automatic Reference Linking by means of MR Lookup. In: Workshop on Linking ans Searching in Distributed Digital Libraries (2002)

[38] LANDAU, Gad M. ; VISHKIN, Uzi: Fast String Matching with k Differences. In: J. Comput. Syst. Sci. 37 (1988), Nr. 1, S. 63–78

[39] LAWRENCE, Steve ; GILES, C. L. ; BOLLACKER, Kurt D.: Digital Libraries and Autonomous Citation Indexing. In: IEEE Computer 32 (1999), Nr. 6, S. 67–71

[40] LEVENSHTEIN, A.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In: Soviet Physics Doklady Bd. 10, 1966, S. 707–710

[41] LEY, Michael: DBLP - Some Lessons Learned. In: PVLDB 2 (2009), Nr. 2, S. 1493–1500

[42] LEY, Michael: DBLP Bibliography - Home Page. http://www.informatik.uni-trier.de/~ley/db/, 2011

[43] LI, Huajing ; COUNCILL, Isaac G. ; BOLELLI, Levent ; ZHOU, Ding ; SONG, Yang ; LEE, Wang-Chien ; SIVASUBRAMANIAM, Anand ; GILES, C. L.: CiteSeer$^{\chi}$: a scalable autonomous scientific digital library. In: Infoscale, 2006, S. 18

[44] MCCALLUM, Andrew ; NIGAM, Kamal ; RENNIE, Jason ; SEYMORE, Kristie: Automating the Construction of Internet Portals with Machine Learning. In: Inf. Retr. 3 (2000), Nr. 2, S. 127–163

[45] NEEDLEMAN, S. B. ; WUNSCH, C. D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. In: Journal of molecular biology 48 (1970), Nr. 3, S. 443–453

[46] OKADA, Takashi ; TAKASU, Atsuhiro ; ADACHI, Jun: Bibliographic Component Extraction Using Support Vector Machines and Hidden Markov Models. In: ECDL, 2004, S. 501–512

[47] OTTMANN, Thomas ; WIDMAYER, Peter: Algorithmen und Datenstrukturen, 4. Auflage. Spektrum, 2002 (Spektrum Lehrbuch). – I–XVII, 1–696 S. – ISBN 978–3–8274–0110–6

[48] PENG, Fuchun ; MCCALLUM, Andrew: Accurate Information Extraction from Research Papers using Conditional Random Fields. In: HLT-NAACL, 2004, S. 329–336

[49] SMITH, T F. ; WATERMAN, M S.: Identification of common molecular subsequences. In: Journal of Molecular Biology 147 (1981), Nr. 1, 195–197. http://www.ncbi.nlm.nih.gov/pubmed/7265238

[50] VILARINHO, Eli Cortez C. ; SILVA, Altigran S. ; GONÇALVES, Marcos A. ; SÁ MESQUITA, Filipe de ; MOURA, Edleno S.: FLUX-CIM: flexible unsupervised extraction of citation metadata. In: JCDL, 2007, S. 215–224

[51] VITERBI, Andrew J.: Viterbi algorithm. In: Scholarpedia 4 (2009), Nr. 1, S. 6246

[52] YIN, Ping ; ZHANG, Ming ; DENG, Zhi-Hong ; YANG, Dongqing: Metadata Extraction from Bibliographies Using Bigram HMM. In: ICADL, 2004, S. 310–319

Beispielverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Hinweise zur Benutzung der Anwendung

Die im Kapitel 6 vorgestellte Anwendung ist unter der URL

`http://stromboli.informatik.uni-freiburg.de:6222/icecite`

zu finden. Die Anmeldung ist entweder mit den Anmeldedaten des Benutzerkontos der Technischen Fakultät der Albert-Ludwigs-Universität in Freiburg oder mit einem zu Testzwecken eingerichteten Gast-Zugang möglich. Für den Gast-Zugang ist der Benutzername `guest` einzugeben und das Textfeld für das Passwort frei zu lassen.
Es wird aber ausdrücklich darauf hingewiesen, dass sich die Anwendung noch in einer frühen Beta-Phase befindet, also mit Fehlern bei der Bedienung der Anwendung zu rechnen ist. Auch ist es möglich, dass Teilfunktionalitäten vorübergehend deaktiviert sind, oder eine der Server vorübergehend nicht erreichbar ist. Über das Symbol in der Fußzeile der Anwendung ist die Erreichbarkeit der Server überprüfbar (vgl. Kapitel 6.4.2).

Anhang B

Beispiele

B.1 Beispiele für fehlgeschlagene Titel-Extraktionen

Symposium: Relative Bioactivity of Functional Foods and Related Dietary Supplements

Functional Foods: Delivering Information to the Oncology Nurse[1,2]

Glen T. Cameron[3] and Mugur V. Geana

Missouri School of Journalism, Columbia, MO 65211

ABSTRACT Recent research suggests a beneficial role of nutrition as possible supportive therapy for cancer patients. A national survey of oncology nurses has shown that nutrition-related issues are an important subject discussed during nurse–patient meetings. The authors applied the activation theory of information exposure to oncology nurses in regard to nutrition information. Findings suggest that oncology nurses who consider nutrition important at a personal level tend to discuss nutrition more with patients and to seek more information about nutrition and cancer. Personal rather than professional motives appear to be triggers for the information search. Implications for health care communication professionals are discussed. J. Nutr. 135: 1253–1255, 2005.

KEY WORDS: • *nutrition* • *cancer* • *oncology* • *nurse*

Any research saga that has a social component culminates with the need to communicate the research findings or its implications to the targeted population. In today's world, dominated by the contrast between information overload and our selectivity regarding the media channels we use to gather information, having the right communication tool could prove essential for the success of any research enterprise.

Things get even more complicated when the subject is perceived by the audience as being a sensitive issue. In the United States, 1,285,000 new patients were diagnosed with cancer in 2002. In the same year, 555,000 people perished due to the illness (1).

Nurses, because of their close interaction with patients and patients' perception of the nurse's role (2), are an efficient channel for distributing health messages. For this reason, nurses' perception and opinions of such nutritional products could have an important impact on improvements in diet for patients. Research by Lev and Ovwen (3) demonstrated the

Nursing theorists emphasize the central role of patient–nurse communication (4) and agree on the key role of the nurse in ensuring patient compliance of major impact on both disease outcome and the quality of life of the patient. This is an extremely valuable approach, especially considering the modern diagnostic and therapeutic interventions that encourage an approach to cancer disease as a chronic illness.

Carr-Hill et al. (5) and Cullum (6) suggested that nurses have a significant influence on patient outcomes in areas such as nutrition, patient hygiene, education and rehabilitation, and pain control and management.

METHODS

The Missouri School of Journalism, in collaboration with the School of Nursing at the University of Missouri, conducted, through its specialized research outlet, the Center for Advanced Social Research, a national survey of members of the Oncology Nursing Society (response rate of 64%). The survey data were used to assess the

Beispiel B.1: Diese Publikation enthält Textzeilen, die eine größere Schriftart aufweisen als die Textzeilen des tatsächlichen Titels und zudem länger als der definierte Schwellenwert (20 Zeichen) sind. Da für uns die Schriftgröße eines der Hauptidentifikationsmerkmale des Titels ist, wählt unser Algorithmus die Textzeilen »*Symposium: Relative Bioactivity [...]*« als Titel aus.

Value Iteration*

Krishnendu Chatterjee[1] and Thomas A. Henzinger[1,2]

[1] University of California, Berkeley
[2] EPFL, Switzerland

Abstract. We survey value iteration algorithms on graphs. Such algorithms can be used for determining the existence of certain paths (*model checking*), the existence of certain strategies (*game solving*), and the probabilities of certain events (*performance analysis*). We classify the algorithms according to the value domain (boolean, probabilistic, or quantitative); according to the graph structure (nondeterministic, probabilistic, or multi-player); according to the desired property of paths

Beispiel B.2: Der Titel dieser Publikation wird nicht erkannt, weil er die durch den Schwellenwert erforderliche Mindestlänge (20 Zeichen), die eine Zeichenkette erreichen muss, um als Titel identifiziert werden zu können, unterschreitet.

ANTIMICROBIAL AGENTS AND CHEMOTHERAPY, Jan. 2006, p. 336–343 Vol. 50, No. 1
0066-4804/06/$08.00+0 doi:10.1128/AAC.50.1.336–343.2006

VraSR Two-Component Regulatory System and Its Role in Induction of *pbp2* and *vraSR* Expression by Cell Wall Antimicrobials in *Staphylococcus aureus*†

Shaohui Yin,[1] Robert S. Daum,[1,2,3] and Susan Boyle-Vavra[1]*

Department of Pediatrics,[1] Committee on Microbiology,[2] and Committee on Molecular Medicine,[3] The University of Chicago, Chicago, Illinois 60637

Received 15 June 2005/Returned for modification 8 August 2005/Accepted 18 October 2005

The VraS/VraR two-component system (VraSR) regulates transcriptional induction of penicillin-binding protein 2 (encoded by *pbp2*) by vancomycin in *Staphylococcus aureus*. We have now defined the *vraSR* operon and determined that its induction by β-lactams as well as by vancomycin is autoregulated. Induction of the *pbp2* and *vraSR* operons by β-lactams and related compounds within 1 hour after exposure to the antimicrobials was dependent on *vraS*. However, when a disk diffusion assay that can detect induction of genes over an extended time period was used, induction of the *pbp2* operon was mediated by some β-lactams, including oxacillin; this induction was independent of *vraS*.

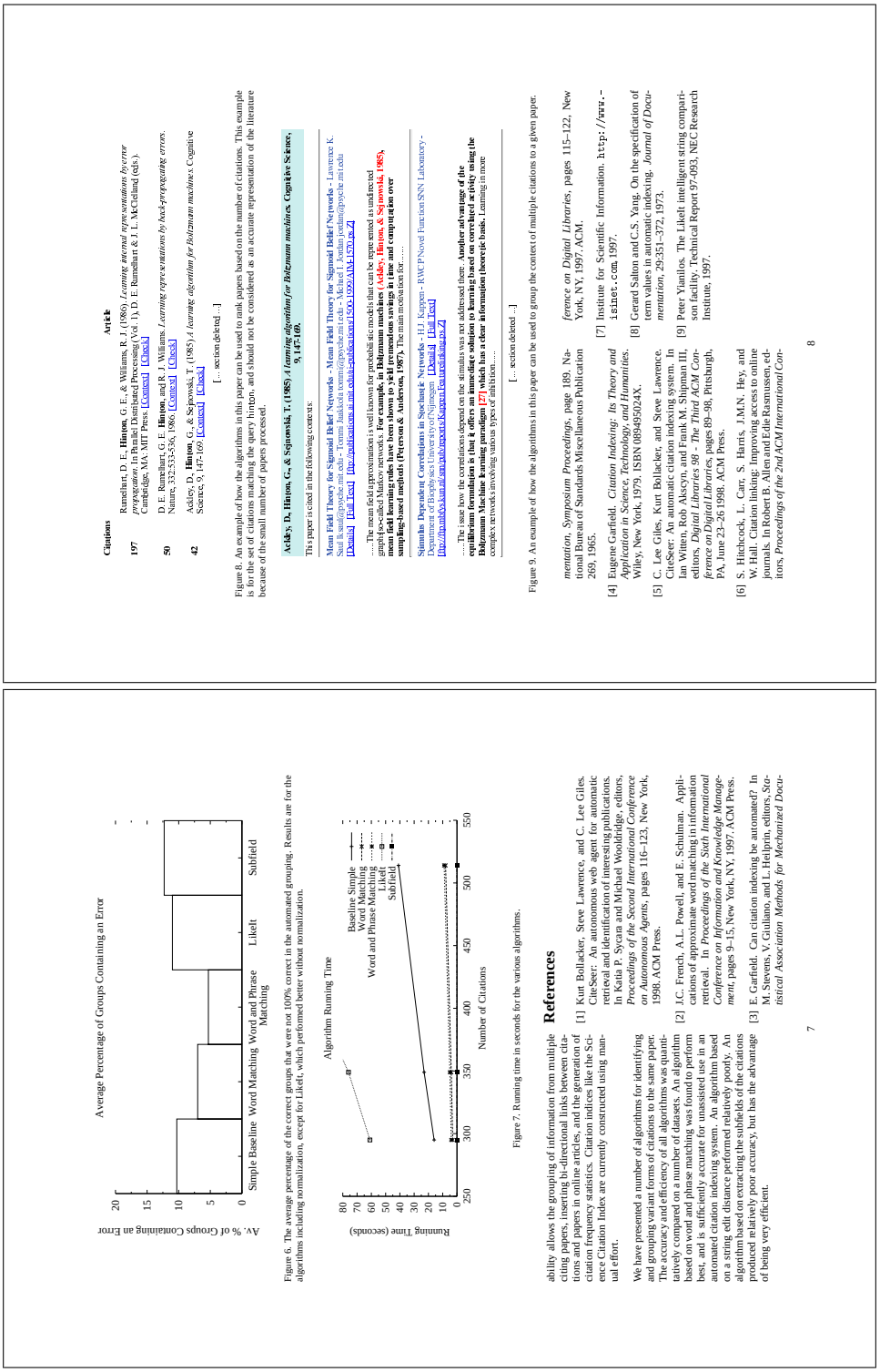
In *Staphylococcus aureus*, penicillin-binding protein 2 (PBP 2) is one of four native penicillin-binding proteins (5, 12, 15, 16, 20). This essential protein plays an accessory role in methicillin and vancomycin (VAN) resistance (3, 4, 11, 14, 16–18). We previously demonstrated that *pbp2* transcription is inducible by

To disrupt the *vraS* gene in the chromosome of strain RN4220, a *vraS* allelic replacement vector was constructed (pVRASR-erm-pCL52.1 [Table 1]). This consisted of a chimera between the *S. aureus* allelic replacement vector, pCL52.1 (10), and pVRASR-erm (Table 1). The latter is a

Beispiel B.3: Die Textzeile »*Staphylococcus aureus*« ist durch eine Kursiv-Schriftart gegenüber den restlichen Titel-Textzeilen hervorgehoben. Veränderungen in der Schriftauszeichnung haben aber zur Folge, dass wir die entsprechenden Textzeilen nicht als zusammenhängendes Element betrachten. Der Titel dieser Publikation wird deshalb nicht korrekt (in diesem Fall nicht vollständig) extrahiert.

Beispiel B.4: Die Titel-Textzeilen dieser Publikation weisen unterschiedliche Schriftgrößen auf. Wie in Beispiel B.3 haben auch Veränderungen in der Schriftgröße zur Folge, dass wir die Textzeilen nicht als zusammenhängendes Element betrachten. Auch dieser Titel wird deshalb nicht vollständig extrahiert.

B.2 Beispiele für fehlgeschlagene Referenzen-Extraktionen



Beispiel B.5: Wenn das Literaturverzeichnis Einschübe (wie z.B. »Figure 8« und »Figure 9« im Beispiel) enthält, hat unser Algorithmus die Referenzen nach diesen Einschüben meist nicht mehr als solche erkannt und das Ende des Literaturverzeichnisses prognostiziert. Die Extraktion der Referenzen war damit unvollständig.

B.3 Weitere Beispiele

```
a             besides      first          its          of           sincere
about         between      five           itself       off          single
above         beyond       fix            journal      often        six
acm           bill         fixed          journals     on           sixty
across        both         for            keep         once         smart
action        bottom       formal         large        one          so
active        but          former         last         only         soc
after         by           formerly       latter       onto         some
afterwards    can          forty          latterly     optimal      somehow
again         cannot       found          learn        or           someone
against       cant         four           learning     other        something
all           chi          from           learns       others       sometime
alg           chapter      front          least        otherwise    sometimes
almost        co           full           less         our          somewhere
alone         con          further        library      ours         springer
along         conf         get            libraries    ourselves    star
already       conference   give           linear       out          still
also          could        go             lncs         over         such
although      couldnt      had            log          own          take
always        cry          has            logs         page         tec
am            de           hasnt          low          pages        ten
among         describe     have           ltd          part         test
amongst       design       he             made         pattern      testing
amoungst      detail       hence          many         per          tests
amount        dev          her            math         perhaps      text
an            do           here           may          please       than
and           done         hereafter      me           power        that
annual        down         hereby         meanwhile    press        the
another       due          herein         might        problem      their
any           during       hereupon       mill         problems     theory
anyhow        each         hers           mine         proc         them
anyone        early        herself        more         proceeding   themselves
anything      ecir         high           moreover     proceedings  then
anyway        eg           higher         most         processing   thence
anywhere      eight        him            mostly       processings  there
appear        either       himself        move         put          thereafter
are           eleven       his            much         random       thereby
around        else         hoc            multi        rather       therefore
art           elsewhere    how            must         recognition  therein
article       empty        however        my           report       thereupon
articles      enabling     human          myself       res          these
as            enough       hundred        name         sat          they
association   etc          i              namely       same         thick
at            europe       ie             neither      sci          thin
aware         even         ieee           netw         see          third
back          event        if             never        seem         this
based         events       image          nevertheless seemed       those
be            ever         images         new          seeming      though
became        every        in             news         seems        three
because       everyone     inc            next         serious      through
become        everything   indeed         nine         set          throughout
becomes       everywhere   inf            no           several      thru
becoming      except       ing            nobody       she          thus
been          exp          int            none         should       time
before        few          interest       noone        show         times
beforehand    fifteen      international  nor          side         tiny
behind        fify         into           not          sigir        to
being         fill         is             now          sigmod       together
below         find         issues         nowhere      simple       too
beside        fire         it             object       since        [...]
```

Beispiel B.6: Liste von Stoppwörter, die nicht vom Invertierten Index indiziert werden, weil sie im Allgemeinen in Texten sehr häufig auftreten und deshalb irrelevant für eine aussagekräftige Suche sind.