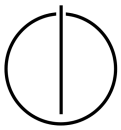
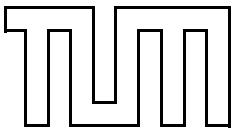


Technische Universität München
Fakultät für Informatik



Lehrstuhl für Effiziente Algorithmen

Dublettenerkennung
Ähnlichkeitsmaße und Verfahren

Stefan Pfingstl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Bernd Radig
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Ernst W. Mayr
2.	Univ.-Prof. Dr. Johann Schlichter

Die Dissertation wurde am 14. März 2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 04. September 2007 angenommen.

Document Classification according to ACM CCS (1998)

Categories and subject descriptors:

`F.2.m` [Analysis of Algorithms and Problem Complexity]:
Miscellaneous

`H.3.3` [Information Storage and Retrieval]:
Information Search and Retrieval—Clustering

`H.3.7` [Information Storage and Retrieval]:
Digital Libraries

`I.5.3` [Pattern Recognizion]:
Clustering—Algorithms, Similarity measures

General Terms: Data cleansing, Duplicate elimination, Record matching

Zusammenfassung

In großen, ständig wachsenden bibliographischen Datenbanken kann es nicht ausgeschlossen werden, dass einzelne Artikel oder sogar komplette Zeitschriften doppelt erfasst werden. Diese Dubletten zu erkennen erfordert zum einen geeignete Methoden, die entscheiden, ob zwei Datensätze identisch sind, und zum anderen Verfahren, die die Anzahl der nötigen Vergleiche minimieren, um diese Dubletten zu finden. In dieser Arbeit wird das Sliding Window* Verfahren vorgestellt, eine Erweiterung des Sliding Window Verfahrens, das zusätzlich zum Suchfenster weitere Daten aus dem Autorengraphen mit einbezieht. Für die Vergleichsoperation findet der Map-Comparator Anwendung. Mit dieser Kombination ist eine sehr zuverlässige Deduplizierung des Datenbestandes möglich.
Ein ähnliches Problem stellt die Identifizierung der Autoren dar. Durch den Einsatz von Heuristiken und der Ableitung weiterer Daten aus den bestehenden kann mit Hilfe eines iterativen, clusterbasierten Verfahrens sehr zuverlässig entschieden werden, welche Autoren die gleiche Person darstellen.

Danksagung

Die vorliegende Arbeit wäre nicht möglich gewesen, ohne die außergewöhnlich gute und intensive Zusammenarbeit mit Prof. Dr. Ernst W. Mayr.

Mein ganz besonderer Dank gilt meinen Kollegen an der TU München und meinem Bruder, die mir bei allen aufgetretenen Problemen jederzeit sehr hilfreich zur Seite standen.

Insbesondere möchte ich meinen Eltern und meiner Ehefrau Danke sagen, die mir eine Universitätsausbildung ermöglicht und mich in jeder Hinsicht unterstützt haben.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Am Lehrstuhl für effiziente Algorithmen von Prof. Dr. Ernst W. Mayr existiert seit etwa 20 Jahren die bibliographische Datenbank LEABib. In ihr sind mehr als 82.300 Datensätze aus dem Bereich der theoretischen Informatik enthalten. Kontinuierlich werden neue Daten erfasst und in der LEABib veröffentlicht. Bei der Erfassung sind Fehler in den Daten und Doppelterfassungen nicht auszuschließen.
In großen Datenbeständen, die fortlaufend durch neue Daten erweitert werden, besteht im Allgemeinen das Problem, doppelte Datensätze zu erkennen und zu eliminieren. Dabei kann zwischen zwei Anforderungen unterschieden werden: Bereinigen eines bestehenden Datenbestandes oder Erkennen der Dubletten beim Einfügen in einen dublettenfreien Datenbestand. Beide Problemstellungen können teils mit den gleichen Verfahren gelöst werden, wobei allerdings unterschiedliche Hilfsdaten bereitgestellt werden müssen.
Eine Reihe von Anwendungen erlaubt das Bereinigen der Dubletten eines Datenbestandes, z.B. bietet iTunes eine Funktion zum Finden von doppelten MP3's in der Datenbank. Für Bilddateien bietet PictureRelate[1] eine Funktion, die nach ähnlichen Bildern sucht.

Für die Dublettenbereinigung einer Datenmenge wurden verschiedene Verfahren entwickelt. Ziel jeder Methode ist es, mit einer minimalen Anzahl von Vergleichen alle Dubletten im Datenbestand zu finden.
Ein einfaches und zuverlässiges Verfahren ist die vollständige Suche. Sie ermöglicht ein Erkennen aller Dubletten, allerdings auf Kosten der Laufzeit. In [HS95] und [HS98] beschreiben Mauricio A. Hernández und Salvatore J. Stolfo das Sliding Window Verfahren, das, ähnlich wie eine vollständige Suche, das Finden von Dubletten auf ein Fenster der Größe k beschränkt. Allgemein arbeitet dieses Verfahren in drei Schritten: 1. Erzeugen von Schlüsseln, 2. Sortieren der Daten anhand der Schlüssel, 3. Suche nach Dubletten im Fenster der Größe k.
Eine Erweiterung ist das Multipass Sliding Window Verfahren. Hierbei werden die drei Schritte mit jeweils verschiedenen Funktionen zum Erzeugen der Schlüssel verwendet.
Eine Alternative zu obigen Verfahren stellen Cluster-Verfahren dar. Dabei werden die Da-

[1] http://www.walthelm.net/picture-relate/

ten mit Hilfe einer Cluster-Funktion in disjunkte Mengen eingeteilt und die Suche nach Dubletten erfolgt ausschließlich innerhalb dieser Mengen.
Weder das Sliding Window- noch das Cluster-Verfahren ist uneingeschränkt nutzbar zum Finden von Dubletten in bibliographischen Daten, da sich bei der Erfassung der Daten unterschiedliche Fehler einschleichen, so dass beide Verfahren nicht alle Dubletten finden. Durch eine Erweiterung der Verfahren ist es aber möglich, die Fehler in den Daten entweder zu erkennen und zu bereinigen oder zumindest so zu behandeln, dass sie sich nicht auf die Dublettenerkennung auswirken. Eines dieser Hilfsmittel ist der Autorengraph [Mut04]. Im Autorengraphen werden Beziehungen, die durch die Publikationen gegeben sind (z.B. Mitautorenbeziehung), dargestellt. Er ermöglicht es, Fehler in den Daten bei der Dublettensuche zu umgehen.

Neben einem Verfahren, das bei der Suche nach Dubletten die Anzahl der Vergleiche minimiert, ist eine Methode nötig, die für zwei gegebene Daten entscheidet, ob diese identisch oder verschieden sind. Hierzu sind Verfahren nötig, die zum einen zwei Inhalte (in diesem Falle Zeichenketten) auf Ähnlichkeit prüfen können, sowie Algorithmen, die entscheiden, ob zwei Datensätze gleich sind. In bibliographischen Daten sind im Allgemeinen nicht immer alle Felder angegeben, so dass der Vergleich zweier Datensätze diese „Verschiedenheit“ mit berücksichtigen muss.
Um zwei Strings zu vergleichen, existieren unterschiedliche Algorithmen, z.B. der Hamming-Abstand [Ham50] oder die Edit-Distanz [Lev66]. Da die Berechnung von Abstandsfunktionen teils mit hohem Aufwand verbunden ist, ist es wichtig, die Abstandsberechnung zu optimieren und eine untere Schranke des Abstands zu bestimmen, um unnötige Berechnungen zu vermeiden.

Kapitel 2 führt in die LEABib ein. Es werden die verschiedenen Tools zur Erfassung, Korrektur und Veröffentlichung der Daten vorgestellt, sowie die Weboberfläche, über die die Daten der LEABib abgefragt werden können.

Kapitel 3 beschreibt ein Verfahren, das für zwei gegebene Datensätze zuverlässig entscheidet, ob diese identisch sind. Dabei wird ausgehend von einem naiven Verfahren ein Algorithmus entwickelt, der mit Hilfe der Edit-Distanz zwei assoziative Arrays vergleichen kann, wobei die Schlüssel der beiden Arrays nicht identisch sein müssen. Um die Berechnung der Edit-Distanz zu beschleunigen, erfolgt einerseits eine untere Abschätzung der Edit-Distanz [Buh01], um unnötige Berechnungen zu vermeiden, und andererseits eine Optimierung der Distanzberechnung, die bei Vorgabe einer maximalen Fehleranzahl die Berechnung entweder vorzeitig abbricht oder die Anzahl der Berechnungsschritte verringert. Durch diese beiden Optimierungen konnte eine Geschwindigkeitssteigerung um Faktor 2 erreicht werden.

Kapitel 4 zeigt die verschiedenen Verfahren, um die Anzahl der Vergleiche bei der Suche nach Dubletten möglichst gering zu halten. Neben den klassischen Verfahren, Sliding Window [HS95], [HS98] und Clustering [Ber02], wird ein Verfahren vorgestellt, das auch mit Fehlern in den Daten alle Dubletten zuverlässig erkennt. Diese Verfahren lösen zuerst das Problem der Bereinigung des gesamten Datenbestandes. Durch Erweiterungen und entspre-

chende Hilfsdaten werden die Verfahren ergänzt, um auch das Problem der inkrementellen Deduplizierung zu lösen.

Eine weitere Aufgabenstellung, die nicht direkt mit dem Erkennen von Dubletten zusammenhängt, aber eng damit verwandt ist, wird in **Kapitel 5** behandelt. Hier wird ein Verfahren gezeigt, das es ermöglicht, anhand eines Journals oder weniger Artikel ähnliche Journale zu finden. Bei der Suche nach Literatur zu einem bestimmten Thema sind dem Recherchierenden nicht immer alle relevanten Zeitschriften bekannt. Mit der Möglichkeit, nach ähnlichen Zeitschriften zu einem vorgegebenen Thema, das durch einzelne Artikel vorgegeben wird, zu suchen, kann auf diese Weise eine Unterstützung in der Recherche erfolgen.

Kapitel 6 behandelt das Problem der Identifizierung der einzelnen Autoren. Hierbei gilt es, das Problem zu lösen, ob der Autor der Publikation a die gleiche Person darstellt, wie der Autor der Publikation b. Die Schwierigkeit besteht darin, dass bei der Erfassung der Autoren sich unterschiedliche Fehler eingeschlichen haben. Diese müssen bei der Autorenidentifizierung mit berücksichtigt werden. Neben syntaktischen Methoden (Normalisierung der Daten, Bildung der Initialen der Vornamen) findet auch hier wieder der Autorengraph [Mut04] Verwendung. Um die Zuverlässigkeit der Identifizierung weiter zu erhöhen, werden weitere Daten, wie z.B. die Publikationszeiträume abgeleitet und berücksichtigt.

Kapitel 2

LEABib

Die bibliographische Datenbank LEABib[1] des Lehrstuhls für Effiziente Algorithmen der Technischen Universität München wurde etwa 1985 begonnen und wird seitdem kontinuierlich gepflegt und fortgeschrieben. Sie beinhaltet mehr als 82.300[2] bibliographische Informationen aus dem Bereich der theoretischen Informatik. Eine interaktive Webschnittstelle unterstützt den Benutzer bei der Suche nach Publikationen.

2.1 Webschnittstelle der LEABib

Die Daten der LEABib sind unter der URL *http://wwwmayr.in.tum.de/leabib/* über das Internet frei abrufbar. Der Benutzer kann in einem HTML-Formular gezielt nach Daten recherchieren (entweder alle Felder durchsuchen oder nur bestimmte). Desweiteren wird eine fehlertolerante Suche unterstützt.

Die Suche erfolgt mittels *leagrep* [Bar95]. Dieses Tool durchsucht die LEABib-Datei nach den angegebenen Daten und liefert die gefundenen BibTEX-Daten an die Webanwendung zurück, die diese entsprechend für die Anzeige im Browser aufbereitet.

In der Anzeige sind alle relevanten Daten untereinander verlinkt, so dass weitere Anfragen durch einfaches Anklicken eines Links angestoßen werden können. Durch Klick auf einen Autorennamen erhält man eine Liste der Veröffentlichungen des Autors, wählt man die Ausgabe eines Journals, so wird das entsprechende Inhaltsverzeichnis aufgelistet. Auch sind die Errata mit ihren Original-Publikationen verlinkt, so dass es auf einfache Weise möglich ist, vom Erratum zum Original zu gelangen und umgekehrt. Jede Anzeige der Daten im Browser kann in unterschiedlichen Formaten ausgegeben werden (z.B. BibTEX, PDF, ...).

[1] http://wwwmayr.in.tum.de/leabib/
[2] Stand: September 2006

Abbildung 2.1: LEABib

2.2 Datenerfassung

Die Erfassung der Daten erfolgt zum Teil per Hand, größtenteils aber halbautomatisch mit Hilfe des Wrappertools DataGen. Mit Hilfe von DataGen ist es möglich, bibliographische Daten aus Inhaltsverzeichnissen im HTML-Format zu extrahieren. Geeignete Skripte konvertieren die extrahierten Daten ins BibTEX-Format, bereiten mathematische Formeln auf, behandeln die Angabe der Autorennamen und ersetzen die Sonderzeichen durch LATEX-Kodierungen. Nach der Erfassung erfolgt eine manuelle Korrektur der Daten mit dem io-port.net-Editor.

2.2.1 DataGen

Mit DataGen ist es möglich, aus Inhaltsverzeichnissen im HTML-Format (oder auch anderen Formaten) bibliographische Daten zu gewinnen. Entsprechende Wrapper extrahieren die relevanten Daten, und die nachgeschalteten Skripten korrigieren und passen die gewonnenen Daten an die BibTEX-Schreibweise an.
Im Tool DataGen kommen HLRT-Wrapper [Kus00] zum Einsatz. Ein Wrapper besteht jeweils aus einer Start- und Enderegel. Diese beiden Regeln geben an, wie der Beginn und das Ende eines Datums im Datensatz gefunden wird. Eine Start- und Enderegel für das gesamte Dokument definieren den zu durchsuchenden Seitenbereich. Eine weitere Elementregel definiert die Größe eines Datensatzes, in dem die einzelnen Daten extrahiert werden sollen.

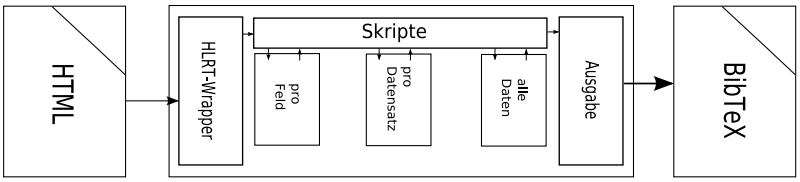


Abbildung 2.2: DataGen

Die durch den Wrapper automatisch gewonnenen Daten sind nicht immer fehlerfrei. Bei dieser Vorgehensweise treten folgende Fehlerquellen auf:

- *Autorennamen:* Die Autorennamen sollen in einer einheitlichen Schreibweise erfasst werden. Die verschiedenen Verlage verwenden unterschiedliche Arten, um die einzelnen Autoren einer Publikation anzugeben und zu trennen (z.B. `,` `<br>` `\n`), wobei in BibTeX diese immer mit „ and “ getrennt werden sollen. Ebenso ist die Angabe der einzelnen Autoren nicht einheitlich. Diese werden automatisch in die Form <Nachname>, <Vorname> gebracht und die Namenszusätze entsprechend den Regeln eingefügt (siehe [Kir]).

- *Mathematische Formeln:* In den HTML-Seiten werden mathematische Formeln meist als Grafiken oder im MathML-Format eingebunden. Diese Formate müssen durch entsprechende Skripte erkannt, entfernt und als LaTeX-Code neu eingefügt werden. Diese Ersetzung kann nur für einen Teil der Formeln automatisch erfolgen, da es nicht immer möglich ist, aus Grafiken den LaTeX-Code der Formel zu rekonstruieren. Einige Verlage geben im „alt“ oder „title“-Attribut des „img“-Tag der Grafik den LaTeX-Code an. Ebenso kann mit Hilfe von XSL Transformationen aus MathML-Code LaTeX-Code erzeugt werden. Auf diese Weise lässt sich die Mehrheit der Formeln automatisch nach LaTeX konvertieren.

- *Sonderzeichen:* Sonderzeichen in HTML oder als Grafiken gilt es, nach LaTeX zu übersetzen (z.B. ä nach \”a, `<img src=''15242.gif'' alt=''eacute''>` `nach` \'e, ...). Zuordnungstabellen, in denen die entsprechenden LaTeX-Codes hinterlegt sind, erlauben eine einfache und zuverlässige Umwandlung dieser Sonderzeichen nach LaTeX.

- *Weitere HTML-Konstrukte:* Die verbleibenden HTML-Konstrukte müssen nach LaTeX übersetzt oder entfernt werden (z.B. Text nach {\bf Text}). Durch Suchen und Ersetzen ist es möglich, diese restlichen HTML-Tags zu entfernen.

- *Vordefinierte Schreibweisen:* Für bestimmte Felder gelten vordefinierte Schreibweisen. Die Daten aus den HTML-Seiten müssen an diese Schreibweisen angepasst werden (z.B. Seitenzahlen: '3–5' anstatt '3– –5', Angabe der Journalnamen, ...). Durch hinterlegte Masken, in denen die fest vorgegebenen Daten hinterlegt sind, wird sichergestellt, dass die Angabe der Journal-, Serientitel, Verlage usw. einheitlich erfolgt. Die restlichen Daten werden wiederum mit entsprechenden Skripten aufbereitet.

- *Unvollständige Daten:* Oftmals sind die Daten nicht vollständig angegeben. Meist fehlen Daten wie z.B. das Volume oder die Nummer. Diese Daten müssen anschließend bei der manuellen Korrektur nachgetragen werden. Desweiteren geben die Verlage bei den Seitenzahlen meist nur die Startseite der Artikel an. Mit Hilfe eines Skriptes können die Endeseiten automatisch eingefügt werden, indem als Endeseite des Artikels die Startseite des nächsten Artikels minus 1 gesetzt wird.

Eine Großzahl der Fehler kann mit entsprechenden Skripten automatisch korrigiert werden. Die verbleibenden Fehler werden anschließend mit dem io-port.net-Editor von Hand nachkorrigiert.
Die Herausforderung besteht darin, einen geeigneten Wrapper zu finden, der es ermöglicht, die Daten aus den HTML-Seiten zu extrahieren. Dazu dient das Tool wGenerate [Han03]. Damit ist es möglich, einen Wrapper anhand von Beispielen automatisch zu generieren. Dazu lädt man eine oder mehrere Seiten in das Tool und markiert die zu extrahierenden Daten. Anhand dieser Markierungen versucht das Tool einen Wrapper zu erstellen, der exakt diese markierten Daten extrahiert. Dazu werden in der 1. Phase alle gültigen Regeln für die jeweiligen Daten und die Head- und Tail-Regel generiert. In der 2. Phase erfolgt die Erstellung des Wrappers anhand der gültigen Regeln, wobei die „besten“ Regeln verwendet werden.

2.2.2 io-port.net-Editor

Der io-port.net-Editor wurde im Rahmen des Projektes FIS-I[3] an der TU München entwickelt, um bibliographische Daten schnell und korrekt zu erfassen. Er unterstützt den Benutzer bei der Erfassung und Korrektur von Daten mit geeigneten Hilfsmitteln, wie z.B. einem LaTeX-Formeleditor, Wertelisten für Journal- und Serientitel und automatischen Prüfungen der Daten auf Korrektheit (LaTeX-Schreibweisen und -Klammerung, fehlende Seiten, ...). Bevor die Daten in die LEABib eingefügt werden, erfolgt im Editor eine Prüfung der Daten auf Korrektheit. Dabei werden alle erfassten Felder auf Richtigkeit überprüft, wie z.B. die Angabe der Autorennamen, Titel der Serien und Journale und richtige LaTeX-Codierungen. Fehlerhafte Daten können somit erkannt und korrigiert werden.

2.3 Veröffentlichung von Daten

Nachdem die bibliographischen Daten erfasst, korrigiert und freigegeben wurden, erfolgt die Veröffentlichung der Daten in der LEABib mit Hilfe des Tools jLEABibWizard. Im ersten Schritt erfolgt die Vergabe von Schlüsseln (Citkeys) und anschließend die Veröffentlichung durch Einfügen der neuen Daten in die LEABib-Datei. Neben Einfügen neuer Daten unterstützt das Programm auch das Aktualisieren oder Löschen von bestehenden Daten und das Prüfen der LEABib-Datei auf Gültigkeit. Diese Funktionen stehen sowohl lokal als auch remote über SSH zur Verfügung.

[3] FachInformationsSystem Informatik

Durch den verstärkten Einsatz der Tools zur automatischen Datenerfassung (DataGen) und den io-port.net-Editor konnte die Anzahl der neu erfassten Daten deutlich gesteigert werden, wobei die Qualität dieser Daten ebenfalls erhöht wurde.

```
node:1
key:key
startrule:<body
endrule:</body>
elemrule:index terms
children:2,3,4,5
node:2
key:title
startrule:class=''medium-text'' target=''_self''><strong>
endrule:</strong>
node:3
key:author
startrule:</strong></a><br>
endrule:<br><small>
node:4
key:pages
startrule:<small>,:
endrule:</small>
node:5
key:url
startrule:HREF=''citation.cfm?id=
endrule:&jmp=cit
```

Abbildung 2.3: Wrapper-Definition

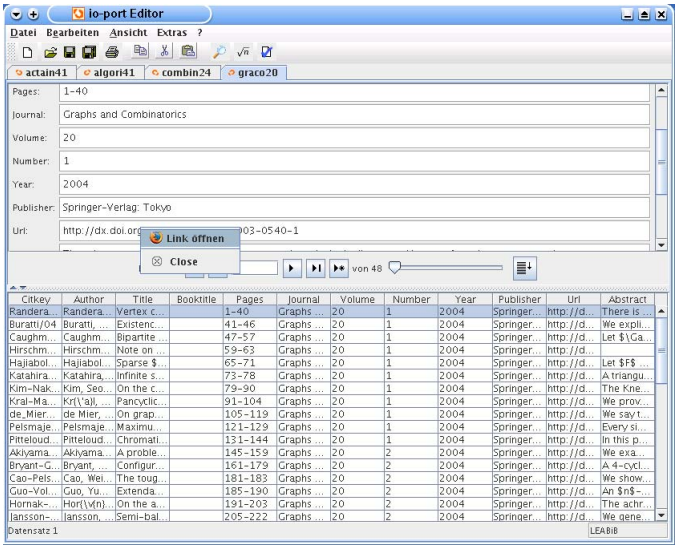


Abbildung 2.4: Der io-port.net-Editor

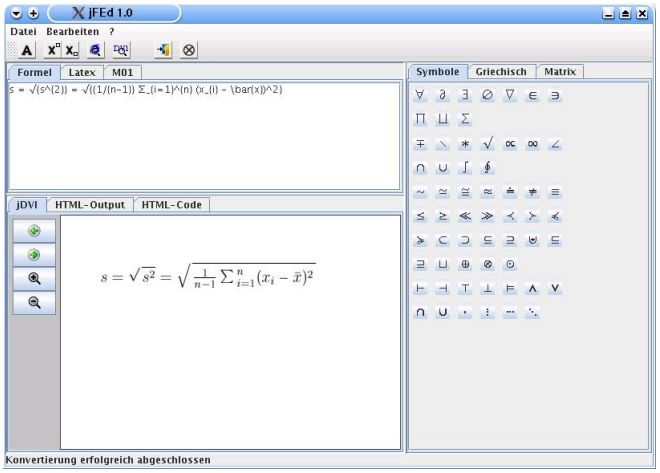


Abbildung 2.5: Der LaTeX-Formeleditor

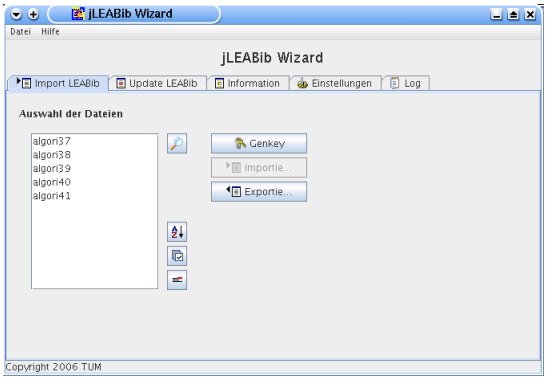


Abbildung 2.6: jLEABibWizard

%Start:
%citkey:Knuth/96
%author:Knuth, Donald E.
%title:An exact analysis of stable allocation
%booktitle:
%journal:J. Algorithms
%pages:431-442
%year:1996
%number:2
%volume:20
%series:
%keywords:
%abstract:
%editor:
%publisher:Academic Press: New York-London-Toronto-Sydney-San Francisco
%institution:
%organization:
%type:
%note:
%pcomment:
%url:
%End:

Abbildung 2.7: LEABib-Datensatz

Kapitel 3

Ähnlichkeitsmaße für bibliographische Daten

Dieses Kapitel gibt einen Überblick, wie bibliographische Daten aufgebaut sind und diese abgelegt werden können, so dass der Zugriff auf die einzelnen Inhalte (Autor, Titel, ...) effizient erfolgen kann. Im Anschluss wird ein Algorithmus entwickelt, der zwei bibliographische Datensätze auf Ähnlichkeit überprüft. Ausgehend von einem naiven Ansatz wird dieser schrittweise verbessert, bis ein Verfahren entstanden ist, das für zwei Datensätze zuverlässig entscheidet, ob es sich um identische Daten handelt.
Neben rein syntaktischen Methoden (Edit-Distanz [Lev66]) werden auch Heuristiken gezeigt, die speziell auf die auftretenden Fehler in den Feldinhalten zugeschnitten sind. Um die Berechnung der Edit-Distanz zu beschleunigen, werden Verfahren gezeigt, die zum einen den Mindestabstand abschätzen [Buh01] und zum anderen den Berechnungsaufwand durch entsprechende Optimierungen gering halten.

3.1 Bibliographische Daten

Für die Erfassung von bibliographischen Daten existieren eine Reihe verschiedener Formate, z.B. BibTEX[Pat88], Dublin-Core[Ini], RIS[Res01], DXF2[1][HLL+03],... Im universitären Umfeld wird überwiegend das BibTEX-Format eingesetzt, da dieses Format automatisch beim Schreiben eines Artikels mit LATEX eingebunden werden kann. Das BibTEX-Format definiert, welche Daten für eine Publikation erfasst werden müssen bzw. sollen und das Dateiformat, in dem die Daten gespeichert werden. Für BibTEX existieren zahlreiche Tools, die den Benutzer bei der Erfassung von bibliographischen Daten unterstützen (io-port.net-Editor, JabRef, BibEdit, ...). Ebenso sind zahlreiche Bibliotheken für die unterschiedlichen Programmiersprachen verfügbar, die einen einfachen Zugriff auf die BibTEX-Daten bereitstellen. Die Java-Bibliothek javabib[2] stellt zum Beispiel die nötigen Klassen und Methoden

[1] Austauschformat www.io-port.net
[2] http://www-plan.cs.colorado.edu/henkel/stuff/javabib/

zur Verfügung, um mit der Programmiersprache Java auf die einzelnen Daten in einer BibTEX-Datei zuzugreifen.

3.2 Assoziative Arrays

In herkömmlichen Arrays (Tabellen) wird für den Zugriff auf die einzelnen Elemente ein Index verwendet, der i. A. bei 0 beginnt. Man kann bei Arrays zwischen statischen und dynamischen unterscheiden. Bei statischen Arrays muss beim Deklarieren die maximale Größe angegeben werden, wobei im Gegensatz bei dynamischen Arrays die Größe während der Laufzeit variieren kann.

Um bibliographische Daten, wie in Abb. 3.1 dargestellt, zu speichern, ist ein Array nicht geeignet, da zum einen die Anzahl der Felder pro Datensatz variiert, zum anderen ein Datum eines Datensatzes aus Feldname und -inhalt besteht. Dieses Problem kann umgangen werden, indem man anstatt normalen Arrays assoziative Arrays verwendet.

Ein assoziatives Array ist eine Spezialform eines Arrays. Bei assoziativen Arrays verwendet man anstatt eines numerischen Index Schlüssel zur Identifizierung der einzelnen Elemente. Die Schlüssel können wie bei normalen Arrays Zahlen darstellen, oder aber auch beliebige Objekte (Strukturen) sein. Die verschiedenen Implementierungen bauen entweder auf Hashtabellen oder Bäumen auf.

Bei bibliographischen Daten bietet sich als Index ein String an, und zwar der Feldname. Unter dem Schlüssel Feldname legt man den entsprechenden Feldinhalt ab.

Assoziative Arrays, die in Java als Maps bezeichnet werden, stellen folgende grundlegende Funktionen für den Zugriff bereit:

- `put(key, value)`: legt den Wert *value* unter dem Schlüssel *key* ab
- `get(key)`: liefert den unter dem Schlüssel *key* gespeicherten Wert
- `getKeys()`: liefert eine Liste der vorhandenen Schlüssel
- `getValues()`: liefert eine Liste der gespeicherten Werte

Definitionen

Für die Berechnung der Ähnlichkeit zweier Datensätze werden im Folgenden die unten angegebenen Definitionen häufiger verwendet.

```
@Book{Knuth97,
   AUTHOR = {Donald E. Knuth},
   TITLE = {Fundamental Algorithms},
   VOLUME = {1},
   SERIES = {The Art of Computer Programming},
   PUBLISHER = {Addison-Wesley},
   ADDRESS = {Reading, Massachusetts},
   PAGES = {650},
   YEAR = {1997},
}

@Article{Chung-Lu/05,
  AUTHOR = {Chung, Kai-Min and Lu, Hsueh-I},
  TITLE = {An optimal algorithm for the maximum-density segment problem},
  JOURNAL = {SIAM J. Comput.},
  VOLUME = {34},
  NUMBER = {2},
  PAGES = {373-387},
  YEAR = {2004-2005},
  EDITOR = {Tardos, E.},
  URL = {http://dx.doi.org/10.1137/S0097539704440430},
  PUBLISHER = {Society for Industrial and Applied Mathematics},
  ADDRESS = {Philadelphia, PA},
}
```

Abbildung 3.1: Zwei bibliographische Datensätze

Σ	Alphabet
$\|\Sigma\|$	Alphabetgröße (Anzahl der Zeichen in Σ)
$c \in \Sigma$	Buchstabe, Zeichen
$c_1 c_2 \dots c_n$ mit $c_i \in \Sigma$	String, Zeichenkette der Länge n
$\|s\|$	Länge der Zeichenkette s
$keys_X$	Menge aller Schlüssel der Map X
$value_X(k)$	Wert des Schlüssels k in der Map X
$d(A, B)$	"Ähnlichkeit" der Maps A und B
	$d(A, B) \rightarrow [0, 1]$
	$d(A, B) = \begin{cases} 0 & \text{keine Ähnlichkeit} \\ 1 & \text{identisch} \end{cases}$

3.3 Ähnlichkeitsmaße für Maps

3.3.1 Naive Methode

Voraussetzung: $keys_A = keys_B$
Der erste naive Ansatz vergleicht für jeden Schlüssel, ob die gespeicherten Werte übereinstimmen. Der Abstand ist definiert als die Anzahl gleicher Werte geteilt durch die Anzahl der Schlüssel.

$$d(A, B) = \frac{|\{value_A(k) = value_B(k) | k \in keys_A\}|}{|keys_A|} \tag{3.1}$$

Diese Definition der Übereinstimmung lässt sich sehr schnell und einfach berechnen. Da für die Daten aber nicht immer die Voraussetzung $keys_A = keys_B$ gilt, kann diese Berechnung i.A. nicht verwendet werden.

3.3.2 Verbesserung 1

Eine erste Verbesserung der naiven Methode besteht darin, alle Felder zu vergleichen, wobei aber nur die Felder betrachtet werden, die in beiden Datensätzen gemeinsam vorhanden sind.
Der Abstand ist folgendermaßen definiert:

$$d(A, B) = \frac{|\{value_A(k) = value_B(k) | k \in S\}|}{|S|} \tag{3.2}$$

wobei $S := keys_A \cap keys_B$.
Diese Definition des Abstandes kann auf alle Datensätze angewandt werden. Bei identischen Daten ist $d(A, B) = 1$ und bei den Daten in Abb. 3.1 ist $d = 0$. Was passiert mit Daten, die keine gemeinsamen Felder besitzen? Diese Daten sollten als verschieden interpretiert werden. Aus diesem Grund definieren wir für diesen Fall $d = \frac{0}{0} = 0$.

```
@Article{Mirwald-Schnorr/92,
  AUTHOR = {Mirwald, R. and Schnorr, C.P.},
  TITLE = {The multiplicative complexity of quadratic boolean forms},
  JOURNAL = {Theor.~Comput.~Sci.},
  VOLUME = {102},
  PAGES = {307-328},
  YEAR = {1992},
  PUBLISHER = {Elsevier Science Publishers B.V.},
}

@Inproceedings{Mirwald-Schnorr/??,
  AUTHOR = {Mirwald, R. and Schnorr, C.P.},
  TITLE = {The multiplicative complexity of quadratic boolean forms},
  BOOKTITLE = {Proceedings of the 28th Annual IEEE Symposium on
               Foundations of Computer Science, FOCS'87
               (Los  Angeles, CA, October 12-14, 1987)},
  ORGANIZATION = {IEEE},
}

@Article{Mirwald-Schnorr/92a,
  AUTHOR = {Mirwald, R. and Schnorr, C.P.},
  TITLE = {The multiplicative complexity of quadratic boolean forms},
  JOURNAL = {Theor.~Comput.~Sci.},
  VOLUME = {102},
  YEAR = {1992},
}

@Article{Craigen/03,
  AUTHOR = {Craigen, R.},
  TITLE = {Boolean and ternary complementary pairs},
  JOURNAL = {J. Comb.~Theory Series A},
  VOLUME = {104},
  NUMBER = {3},
  PAGES = {1-16},
  YEAR = {2003},
}

@Article{Bohler-Reith-Schnoor-Vollmer/05,
  AUTHOR = {B{\"o}hler, Elmar and Reith, Steffen and Schnoor,
            Henning and Vollmer, Heribert},
  TITLE = {Bases for Boolean co-clones},
  JOURNAL = {Inf.~Process.~Lett.},
  VOLUME = {96},
  NUMBER = {2},
  PAGES = {59-66},
  YEAR = {2005},
  EDITOR = {Tarlecki, A.},
}
```

Abbildung 3.2: Bibliographische Datensätze

Ein weiteres Problem dieses Verfahrens wird in Abb. 3.2 gezeigt. Hier ist für die ersten beiden Datensätze $S = \{\text{Author}, \text{Title}\}$ und somit $d(A, B) = \frac{1+1}{2} = 1$. Der Algorithmus bestimmt für diese Daten, dass sie identisch sind, obwohl es sich hier um zwei unterschiedliche Datensätze handelt. Das Problem liegt darin, dass der Algorithmus nur die Feldinhalte der Schlüssel vergleicht, die in beiden Datensätzen gemeinsam enthalten sind.
Für eine alternative Berechnung könnte man S als

$$S = keys_A \cup keys_B$$

definieren. Allerdings werden mit dieser Definition von S viele Dubletten nicht erkannt, da es eine häufige Fehlerquelle ist, dass ein Datensatz „mehr definiert“ ist als ein anderer (siehe Abb. 3.2, erster und dritter Datensatz). Ein Datensatz B ist „mehr definiert“ als A, wenn $keys(A) \subsetneq keys(B)$ gilt.
In obigem Fall sind die Schlüssel $S = \{\text{Author}, \text{Title}, \text{Journal}, \text{Volume}, \text{Pages}, \text{Year}, \text{Publisher}\}$ und $d(A, B) = \frac{1+1+1+1+0+1+0}{7} = \frac{5}{7} \approx 0,71$

Eine weitere Fehlerquelle bei der Suche nach Dubletten verursachen Fehler, die sich bei der Erfassung einschleichen.
Diese Fehler lassen sich in drei Kategorien einteilen:

Rechtschreibfehler sind Fehler, die durch Unkenntnis der korrekten Schreibweise entstehen (nähmlich, Maschiene, Lieble). Sie können durch Verwendung einer Rechtschreibprüfung gefunden und korrigiert werden. Eine Liste häufiger Rechtschreibfehler in englischen Texten ist in [Wik06a] und für deutschsprachige in [Wik06b] zu finden.

Tippfehler sind Fehler, die sich durch Unachtsamkeit einschleichen. Häufige Tippfehler sind: zusätzliche, fehlende oder falsche Zeichen oder Vertauschen von zwei Zeichen (Buchstabendreher) (Abtsand, Zeitng, Zeitunmg). Tippfehler lassen sich ebenfalls meist mit Hilfe einer Rechtschreibprüfung finden und korrigieren. Bestimmte Tippfehler lassen sich aber nicht ohne den Kontext erkennen bzw. korrigieren, z. B. Wein $\leftrightarrow$ Wien. In diesem Fall sind beide Wörter korrekt, aber sie haben verschiedene Bedeutungen. Nur durch Berücksichtigung des Kontexts kann dieser Tippfehler erkannt und korrigiert werden. Da die Daten meist von Hand eingegeben werden, ist die Wahrscheinlichkeit, dass ein Zeichen zuviel oder fehlerhaft eingegeben wurde, um so größer, je näher der Buchstabe dem richtigen Zeichen auf der Tastatur ist.

Formfehler stellen im eigentlichen Sinne keine Fehler dar, aber sie stellen die gleichen Anforderungen an ein Verfahren zur Erkennung von Fehlern. Bei Formfehlern handelt es sich um unterschiedliche Schreibweisen für ein und den selben Inhalt. In LaTeX sind beliebig viele geschweifte Klammern an jeder Stelle erlaubt. Ebenso können mathematische Formeln auf unterschiedliche Weisen angegeben werden. Eine letzte Fehlerquelle stellen die Akzentzeichen dar. Diese können ebenfalls auf verschiedene Arten angegeben werden.

Werden die Feldinhalte nur auf Gleichheit geprüft, so werden geringe Unterschiede in den Inhalten, die durch obige Fehler entstanden sind, nicht berücksichtigt. Um diese Fehler mit einzubeziehen, verwendet man anstatt eines direkten Vergleichs eine Abstandsfunktion auf Zeichenketten.

3.3.3 Hamming-Abstand

Der Hamming-Abstand [Ham50] zweier Zeichenketten a und b ist definiert als die Anzahl der Positionen unterschiedlicher Zeichen in a und b. Da der Hamming-Abstand nur für Zeichenketten mit gleicher Länge definiert ist ($|a| = |b|$), führen wir die Berechnung nur auf den Präfixen $a_1 a_2 \dots a_{\min|a|,|b|}$ und $b_1 b_2 \dots b_{\min|a|,|b|}$ aus und addieren den Längenunterschied $\Big|\ |a| - |b|\ \Big|$ auf:

$$\Big|\ \{i \mid a_i \neq b_i \wedge 1 \leq i \leq \min(|a|,|b|)\}\ \Big| + \Big|\ |a| - |b|\ \Big|)$$

Algorithmus 3.1: Hamming-Abstand

hamming($a[1 \dots n]$, $b[1 \dots m]$)
    $e = |n - m|$

    for $i = 1$ **to** $\min(n, m)$ **do**
        if $a[i] \neq b[i]$ **then** $e = e + 1$
    od

    return e

Der Hamming-Abstand $h(a, b)$ zweier Wörter a und b liegt zwischen 0 und $\max(a, b)$ und kann in linearer Laufzeit und ohne zusätzlichem Speicher berechnet werden.

Beispiel:
hamming(„aber“, „Test“) = 4
hamming(„Algorithmus“, „Algortimus“) = 4

□

In obigem Beispiel kann man sofort erkennen, dass der Hamming-Abstand nicht geeignet ist, um fehlerbehaftete Zeichenketten zu vergleichen. Schon ein einzelner Fehler wird in der Regel einen großen Hamming-Abstand zwischen den beiden Wörtern a und b verursachen. Aus diesem Grund ist es notwendig, eine andere Abstandsfunktion zu verwenden.

3.3.4 Edit-Abstand

In vielen Anwendungen reicht es oftmals nicht aus, zwei Zeichenketten a und b (Strings) nur auf Gleichheit zu prüfen. Da in den Zeichenketten evtl. Fehler enthalten sein können,

liefert ein direkter Vergleich der beiden Zeichenketten entweder *true* (sie stimmen überein) oder $false$ (sie stimmen nicht überein). Auch der Hamming-Abstand liefert kein sinnvolles Ergebnis für diese Problemstellung.
Gesucht ist eine Kenngröße, die angibt, wie ähnlich sich die beiden Zeichenketten sind. Je kleiner diese Kenngröße, desto ähnlicher sind sich die beiden Zeichenketten.
Der Edit-Abstand definiert dazu fünf Operationen, mit Hilfe derer man die Zeichenkette a nach b überführen kann. Dabei arbeitet man das Wort a von links nach rechts ab und konstruiert mit diesen Operationen das Wort b.

Die folgenden Operationen können angewendet werden:

copy c**:** Das Zeichen c von a wird an b angefügt

insert c**:** Das Zeichen c wird an b angefügt

delete c**:** Das Zeichen c wird von a entfernt

change c **to** d**:** Das Zeichen c wird von a entfernt und als d an b angefügt

trans cd**:** Die beiden Zeichen cd von a werden als dc an b angefügt

Der Edit-Abstand gibt die minimale Anzahl von nötigen *insert*, *delete*, *change* und *trans*-Operationen an, um aus a die Zeichenkette b zu erzeugen. Die *copy*-Operation trägt nicht zum Abstand bei.

Der Edit-Abstand definiert einen sinnvollen Abstand, da

- $editdist(a, a) = 0$ (Identität)
- $editdist(a, b) = editdist(b, a)$ (Symmetrie)
- $editdist(a, b) \leq editdist(a, c) + editdist(c, b)$ (Dreiecksungleichung)

Beispiel:
a = abdcdabbd
b = adbadabd

copy a	a
trans bd	adb
change c *to* a	adba
copy d	adbad
copy a	adbada
copy b	adbadab
insert b	adbadabd
copy d	adbadabd

□

Anhand des obigen Beispiels kann man erkennen, das der Edit-Abstand zwischen „abd-cdabbd“ und „adbadabd“ mindestens 3 beträgt.
Die Berechnung der Edit-Distanz zweier Wörter erfolgt im Allgemeinen mittels dynamischer Programmierung.

Algorithmus 3.2: Edit-Abstand

editdist($a[1 \dots n]$, $b[1 \dots m]$)
    $D =$ **new** $Int[0 \dots n][0 \dots m]$
    for $i = 0$ **to** n **do** $D[i][0] = i$ **od**
    for $i = 0$ **to** m **do** $D[0][i] = i$ **od**

    for $i = 1$ **to** n **do**
        for $j = 1$ **to** m **do**
            if $a[i] == b[j]$ **then** $d_1 = D[i-1][j-1]$;
            else $d_1 = D[i-1][j-1] + 1$;
            if $a[i] == b[j-1] \wedge a[i-1] == b[j]$ **then** $d_2 = D[i-2, j-2] + 1$;
            else $d_2 = \inf$;
            $D[i][j] = \min(D[i-1][j] + 1, D[i][j-1] + 1, d_1, d_2)$;
        od
    od

    return $D[n][m]$

Die Tabelle in Abb. 3.1 zeigt die berechneten Werte des obigen Algorithmus. Der Algorithmus füllt die Zellen der Tabelle spaltenweise. Am Ende der Berechnung steht in der rechten unteren Zelle ($D[n][m]$) der Edit-Abstand der beiden Wörter a und b.

		a	d	b	a	d	a	b	d
	0	1	2	3	4	5	6	7	8
a	1	0	1	2	3	4	5	6	7
b	2	1	1	1	2	3	4	5	6
d	3	2	1	1	2	2	3	4	5
c	4	3	2	2	2	3	3	4	5
d	5	4	3	3	3	2	3	4	4
a	6	5	4	4	3	3	2	3	4
b	7	6	5	4	4	4	3	2	3
b	8	7	6	5	5	5	4	3	3
d	9	8	7	6	6	5	5	4	**3**

Tabelle 3.1: Edit-Tabelle

Erweiterung des Edit-Abstands

Die Berechnung des Edit-Abstands benötigt $O(n*m)$ Zeit. In obigem Beispiel werden $8*9 = 72$ Felder der Editiertabelle berechnet (siehe Abb. 3.1). Ist eine maximale Fehleranzahl k bekannt, die nicht überschritten werden darf, so kann die Berechnung des Edit-Abstands beschleunigt und die Laufzeit auf $O(k * m)$ verringert werden.
Nach jeder Iteration wird das Minimum m der aktuell berechneten Spalte bestimmt. Das Minimum m gibt an, wie groß der Edit-Abstand zwischen den beiden Wörtern mindestens ist. Ist dieses m größer als der vorgegebene Abstand k, so kann die Berechnung abgebrochen werden, da der Abstand zwischen den beiden Wörtern die Schranke k übersteigt. Als Rückgabewert der Berechnung kann -1 zurückgegeben werden, um anzuzeigen, dass die maximal zulässige Fehleranzahl überschritten wurde. Zusätzlich zur obigen Optimierung berechnet man die Tabelle nicht mehr spaltenweise, sondern spalten- und zeilenweise. Im Schritt i wird die i-te Zeile und i-te Spalte nur vom Element $i - k$ bis zum Element i berechnet und nur in diesem Teil ($\min(|a|, i)$-te Spalte und $\min(|b|, i)$-te Zeile) das Minimum bestimmt.

Gibt man in obigem Beispiel einen maximalen Fehler von 3 an (der Edit-Abstand der beiden Wörter), so verringert sich die Anzahl der zu berechnenden Felder auf 47 (siehe Abb. 3.2). Bei einer vorgegebenen maximalen Fehleranzahl von 1 reduziert sich der Aufwand auf die Berechnung von nur noch 10 Felder (siehe Abb. 3.3) (bei maximalem Fehler 2 beträgt der Aufwand 34).

		a	d	b	a	d	a	b	d
	0	1	2	3	4	5	6	7	8
a	1	0	1	2	3				
b	2	1	1	1	2	3			
d	3	2	1	1	2	2	3		
c	4	3	2	2	2	3	3	4	
d	5		3	3	3	2	3	4	4
a	6			4	3	3	2	3	4
b	7				4	4	3	2	3
b	8					5	4	3	3
d	9						5	4	**3**

Tabelle 3.2: Edit-Tabelle mit max. Fehler = 3

Abbildung 3.3 zeigt den Zusammenhang der Anzahl nötiger Berechnungen und des Ergebnisses der Bestimmung von $dist$("An approximation algorithm for the TSP","Approximation algorithms for TSP with neighborhoods in the plane", max), bei vorgegebenem max. In diesem Beispiel kann man erkennen, dass bei einer kleinen maximalen Fehleranzahl die Anzahl der Berechnungen klein ist.
Wird diese Definition des Abstandes zweier Wörter gerecht? Betrachtet man die anfangs

		a	d	b	a	d	a	b	d
	0	1	2	3	4	5	6	7	8
a	1	0	1						
b	2	1	1	1					
d	3		1	1	2				
c	4			2	2				
d	5								
a	6								
b	7								
b	8								
d	9								

Tabelle 3.3: Edit-Tabelle mit max. Fehler = 1

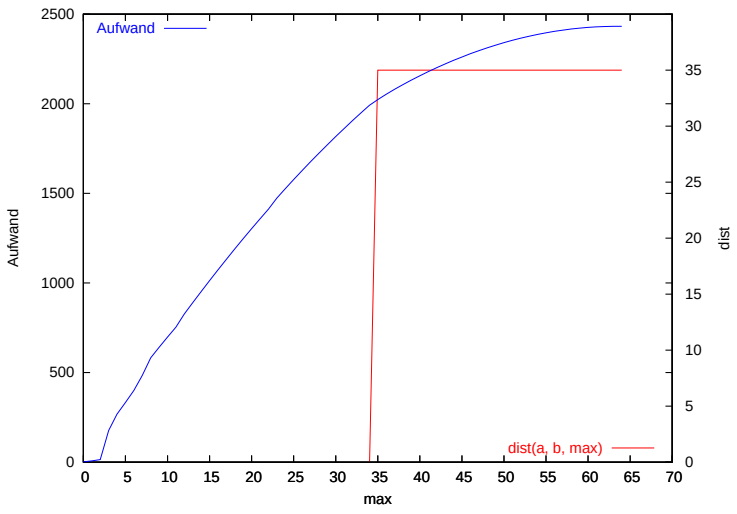


Abbildung 3.3: Berechnung $dist$(„An approximation algorithm for the TSP“,“Approximation algorithms for TSP with neighborhoods in the plane“, max)

erwähnten Fehler (Rechtschreib-, Tipp- und Formfehler) so ist ersichtlich, dass eine einfache Berechnung der Edit-Distanz den auftretenden Tippfehlern nicht gerecht wird. Rechtschreibfehler können durch die Edit-Distanz richtig erkannt werden, aber bei Tippfehlern wird keine Rücksicht genommen, wie wahrscheinlich dieser sein kann. Die Wahrscheinlichkeit, fälschlicher Weise eine benachbarte Taste zu drücken ist höher, als eine weit entfernte. Diese Erkenntnis sollte in der Berechnung mit berücksichtigt werden. Um dies in der Berechnung mit einzubeziehen, muss bei jeder *change*-Operation der Abstand der Tasten auf der Tastatur berücksichtigt werden. Dazu werden die konstanten Kosten der *change*-Operation durch eine Funktion $c_1(a, b) : Character \times Character \rightarrow [0, 1]$ ersetzt. Je größer dieser Abstand der beiden Buchstaben `a` und `b` auf der Tastatur ist, desto höher sind die Kosten des Fehlers. Um diese Abstände effizient berechnen zu können, bietet es sich an, ein Array zu erstellen, in dem für jedes Zeichen c die Position (x_c, y_c) auf der Tastatur eingetragen ist. Der Abstand kann dann mit $\alpha * \max\{|x_a - x_b|, |y_a - y_b|\}$ in konstanter Zeit bestimmt werden (vgl. Maximum- bzw. Tschebyschow-Distanz [Bro05]). Die restlichen Tippfehler sind durch die Opertionen *insert*, *delete* und *trans* abgedeckt.

Abbildung 3.4: Tastaturlayout

Um Formfehler ebenfalls mit einzubeziehen, ist eine Änderung in den Kosten der *insert* und *delete*-Operation nötig. Diese Kosten werden ebenfalls durch eine Funktion $c_2(a, b) : Character \times Character \rightarrow [0, 1]$ ersetzt. Hierbei wird für jedes Zeichen $c \in \Sigma$ angegeben, wie hoch die Kosten für das Einfügen oder Löschen des Zeichens c sind. Um die Formfehler in dieser Tabelle zu berücksichtigen, werden die Kosten für die Zeichen, die Formfehler darstellen, auf 0 gesetzt, für die restlichen auf 1.
Eine andere Vorgehensweise, die Formfehler mit einzubeziehen, ist folgende: Bevor der Edit-Abstand berechnet wird, erfolgt eine „Normalisierung“ der Wörter, das bedeutet, dass alle Sonderzeichen, geschweifte Klammern, Akzentzeichen usw. aus den Wörtern entfernt werden. Für die resultierenden Wörter wird anschließend der Edit-Abstand berechnet. Eine Normalisierung von LaTeX-Texten kann schnell und einfach bestimmt werden. Der Vorteil der Normalisierung ist, dass diese vorab berechnet werden kann und nicht erst bei der Berechnung der Edit-Distanz erfolgen muss.

Abschätzung des Edit-Abstands

Die Berechnung der Edit-Distanz erfordert einen Zeitaufwand von $O(k * n)$, wobei k die maximale Fehleranzahl und n das Maximum der beiden Stringlängen bezeichnet. Sind sehr viele Berechnungen der Edit-Distanz erforderlich, ist es sinnvoll, vor der Berechnung zuerst

zu testen, ob der Edit-Abstand der beiden Zeichenketten kleiner als k sein kann.
Sei $\delta_l = |length(a) - length(b)|$. Der Edit-Abstand zweier Zeichenketten a und b kann durch

$$\delta_l \leq editdist(a, b) \leq hamming(a, b)$$

abgeschätzt werden, das bedeutet, wenn der Längenunterschied der beiden Zeichenketten a und b mehr als k beträgt, so ist eine Berechnung der Edit-Distanz nicht nötig, da diese nicht kleiner als k sein kann (es müssen mindestens δ_l Zeichen entweder eingefügt oder gelöscht werden, um die beiden unterschiedlichen Stringlängen anzupassen).

Locality-Sensitiv Hashing [Buh01] Wir betrachten zwei Strings a und b über dem Alphabet Σ, mit durchschnittlicher Länge l. Wir wählen ein $k < l$ und zufällig und gleichverteilt aus der Menge $\{1, \ldots, d\}$ k Elemente aus und erhalten somit $\{i_1, \ldots, i_k\}$.
Darauf definieren wir eine Funktion $f : \Sigma^l \to \Sigma^k$ wie folgt:

$$f(x) = x[i_1]x[i_2] \ldots x[i_k]$$

Diese Funktion f wird als Locality-Sensitiv Hashing Funktion bezeichnet. Mit f kann in sublinearer Zeit bestimmt werden, ob zwei Strings a und b identisch sind oder nicht. Gilt $f(a) \neq f(b)$, so sind die beiden Strings unterschiedlich. Des weiteren gilt: [Buh01]

$$Pr[f(a) = f(b)] \geq \left(1 - \frac{k}{l}\right)^k \text{ falls } a = b$$

Wie kann mit dieser Funktion die Edit-Distanz abgeschätzt werden? Lässt man in der Berechnung der Edit-Distanz nur die *change*- und *trans*-Operation zu, so kann man die Edit-Distanz durch

$$\frac{hamming(f(a), f(b))}{2} \leq editdist(a, b)$$

abschätzen.

Wie kann nun obiges Verfahren erweitert oder abgeändert werden, um auch die *insert*- und *delete*-Operation zu berücksichtigen? Anstatt feste Positionen in beiden Strings zu vergleichen, zählt man die Vorkommen der einzelnen Buchstaben und versucht daraus eine Abschätzung der Edit-Distanz zu erhalten.
Diese Abschätzung basiert darauf, dass bestimmte Zeichen häufiger vorkommen als andere. In der LEABib treten die Zeichen '␣', 'e', 'a' und 'n' am häufigsten auf (siehe Abb 3.5). Durch Auswahl einer großen Menge von Beispieltexten kann man für eine Sprache die Verteilung der Buchstaben ermitteln. Die Abb 3.5 zeigt die Verteilungen für die Daten in der LEABib, deutsche und englische Beispieltexte. Die Beispieltexte für Deutsch und Englisch wurden aus Internetseiten von renommierten Verlagen (Spiegel, FAZ, Times, New York Times, ...) extrahiert, so dass die Texte für jede Sprache mindestens 250.000 Buchstaben umfassten. Bei allen drei untersuchten Texten sind die am häufigsten vorkommenden Zeichen '␣', 'e', 'a' und 'n' (siehe Kap. A.1).

Wie kann diese Verteilung verwendet werden, um die Edit-Distanz abzuschätzen?

Zählt man in beiden Zeichenketten die Anzahl aller vorkommenden Zeichen, so erhält man einen Fingerprint für jedes der beiden Zeichenketten der für jeden Buchstaben im Alphabet angibt, wie oft dieser Buchstabe in der Zeichenkette vorhanden ist. Anhand der Fingerprints für die Zeichenketten a und b lässt sich bestimmen, in wie vielen Zeichen sich a und b unterscheiden.

Sei $\delta_c = \sum_{i \in \Sigma} |\#_{i_a} - \#_{i_b}|$ die Anzahl der in a und b unterschiedlichen Zeichen und δ_l der Längenunterschied von a und b.

Es gilt:

$$\delta_l \leq \frac{\delta_c + \delta_l}{2} \leq editdist(a, b) \leq hamming(a, b)$$

Die Edit-Distanz kann mit einer *change*-Operation zwei unterschiedliche Zeichen „anpassen“ und mit einer *insert*- oder *delete*-Opertion die Länge einer Zeichenkette ändern. Da sich die Längen der beiden Zeichenketten um $delta_l$ unterscheiden, sind mindestens $delta_l$ *insert*- oder *delete*-Operationen nötig. Jede dieser Operationen ändert δ_c um 1. Die $\delta_c - \delta_l$ verbleibenden unterschiedlichen Zeichen müssen zumindest mit einer *change*-Operation geändert werden und da eine *change*-Operation ein Zeichen c in ein Zeichen d überführt, sind mindestens $\frac{\delta_c - \delta_l}{2}$ *change*-Operationen nötig. Die *tans*-Operation erhöht zwar den Editabstand der beiden Wörter, ändert aber nicht die Werte von δ_c oder δ_l. Fasst man diese beiden Teile zusammen, so erhält man als Abschätzung der Edit-Distanz folgenden Term:

$$editdist(a, b) \geq \frac{\delta_c - \delta_l}{2} + \delta_l = \frac{\delta_c + \delta_l}{2}$$

Der Aufwand zur Berechnung der Fingerprints der beiden Zeichenketten beträgt $O(|\Sigma| + n))$, wobei die Längen der beiden Zeichenketten durch $O(n)$ beschränkt sind.

Bei kleiner Alphabetgröße ist es sinnvoll, alle Zeichen in den Fingerprint aufzunehmen, da der Aufwand der Berechnung des Fingerprints hierbei durch $O(n)$ gegeben ist. Ist $|\Sigma|$ sehr groß ($\Sigma \geq O(n)$, |UTF-Zeichensatz| $= 2^{16} = 65536$), so ist die Erstellung des Fingerprints immer noch durch $O(n)$ beschränkt (da $|\Sigma|$ konstant ist), aber die Berechnung von δ_c wird durch $O(\Sigma)$ bestimmt. In diesem Fall ist es sinnvoll, nicht alle Zeichen aus Σ im Fingerprint zu berücksichtigen, sondern nur die Zeichen, die am häufigsten auftreten. In Abb. 3.5 ist die Verteilung der Buchstaben für die LEABib, deutsch und englisch dargestellt. Verwendet man zur Bestimmung des Fingerprints nur die Zeichen *␣, e, a, n, o, i, r, s, t*, so zählt man im Durchschnitt jeweils ca. 46% der Buchstaben in den beiden Zeichenketten. Durch diese Einschränkung bleibt die Gültigkeit der obigen Abschätzung erhalten, die Genauigkeit aber sinkt.

Beispiel:

$\Sigma = \{a, b, c\}$
$a =$ ababbba und $b =$ babcbbabc
Fingerprint von $a = (3, 4, 0)$ und von $b = (2, 5, 2)$
$\delta_c = |3-2| + |4-5| + |0-2| = 4$ und $\delta_l = \Big|\, |a| - |b| \,\Big| = |7-9| = 2$
$\Rightarrow \mathit{editdist}(\text{ababbba}, \text{babcbbabc}) \geq \frac{4+2}{2} = 3$

□

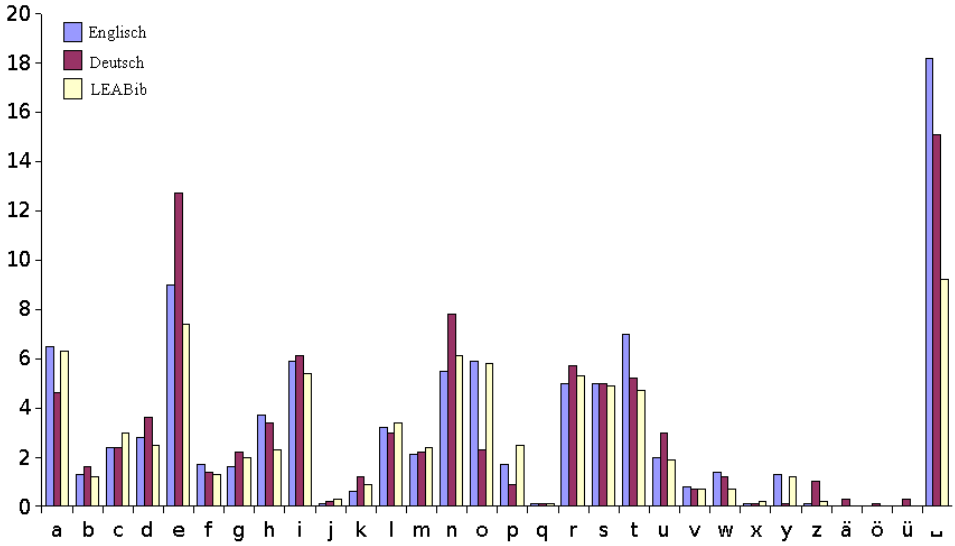


Abbildung 3.5: Buchstabenverteilung

Im Fall der LEABib ist das Alphabet durch den ASCII-Zeichensatz beschränkt und damit ist $|\Sigma| = 256$. Bei dieser Alphabetgröße ist es vorteilhaft, den Fingerprint über dem kompletten Alphabet zu bestimmen, da hier die Laufzeit um Faktor 0.50 geringer ist, als ohne Optimierungen. Legt man den Fingerprint nur über den Zeichen ␣, *e, a, n, o, i, r, s, t* an, so verringert sich die Laufzeit nur um Faktor 0.63 (siehe Kap A.2).

Edit-Abstand und Ähnlichkeit von Zeichenketten

Der Edit-Abstand definiert einen Abstand zweier Zeichenketten. Sind zwei Zeichenketten a und b ähnlich, so ist der Edit-Abstand $\ell(a, b)$ klein. Auf dieser Basis ist es möglich, mit Hilfe des Edit-Abstands eine Ähnlichkeitsrelation anzugeben.
Der Abstand zweier Zeichenketten a und b kann maximal $\max(|a|, |b|)$ betragen, da zuerst alle $\min(|a|, |b|)$ Zeichen mit einer *change*-Operation und anschließend die restlichen $\max(|a|, |b|) - \min(|a|, |b|)$ Zeichen mittels *delete*- oder *insert*-Operation „angepasst“ werden können.

Damit ergibt sich folgende Definition einer Ähnlichkeitsrelation:

$$dist(a,b) = \frac{\max(|a|,|b|) - \ell(a,b)}{\max(|a|,|b|)} = 1 - \frac{\ell(a,b)}{\max(|a|,|b|)} \tag{3.3}$$

3.3.5 Verbesserung 2

Sei

$\ell(a,b)$ = Edit-Abstand der beiden Wörter a und b
$dist(a,b)$ = $1 - \frac{\ell(a,b)}{\max\{|a|,|b|\}}$
S = $keys(A) \cap keys(B)$

$$\frac{\sum_{k \in S} dist(value_A(k), value_B(k))}{|S|} \tag{3.4}$$

Dieses Verfahren kann mit Fehlern, die sich bei der Erfassung der Daten einschleichen, umgehen. Tippfehler werden erkannt und in die Berechnung des Abstands mit eingerechnet. Zwei identische Datensätze, die sich nur durch einen oder wenige Tippfehler unterscheiden, erhalten einen Wert von ≈ 1. Allerdings hat dieses Verfahren immer noch das Problem mit der fehlerhaften Behandlung der ersten beiden Datensätze in Abb. 3.2. Auch hier werden diese beiden als identisch betrachtet, obwohl es sich um zwei verschiedene Daten handelt.

3.3.6 Map-Komparator

Um das letzte Problem zu umgehen, ist eine Betrachtung aller Felder und deren Inhalte nötig. Aber wie sollen Felder und deren Inhalte beurteilt werden, wenn das Feld nur in einem den beiden Datensätze existiert? Hier kann zwischen zwei Fehlerquellen unterschieden werden:

- einer der beiden Datensätze ist mehr definiert als der andere, das heißt, dass die Schlüssel des einen Datensatzes eine Teilmenge der Schlüssel des anderen Datensatzes bilden.

  $$keys_A \subset keys_B \vee keys_B \subset keys_A$$

  Zum Beispiel definiert Datensatz a das JOURNAL, das VOLUME und die NUMBER, Datensatz b nur JOURNAL und VOLUME. In diesem Fall ist die fehlende NUMBER kein Indiz dafür, dass es sich um zwei unterschiedliche Datensätze handelt, sondern dass der Datensatz a mehr definiert ist als b und somit die Bewertung der fehlenden NUMBER geringer eingestuft wird.

- die beiden Mengen der Schlüssel der Datensätze sind keine Teilmengen, das bedeutet, dass die beiden Datensätze evtl. unterschiedliche Publikationstypen beschreiben. Jeder Publikationstyp definiert die Felder, die bei der Erfassung der Daten mindestens angegeben werden sollten. Ist in einem Datensatz das Feld JOURNAL und in einem anderen SERIES angegeben, so handelt es sich wahrscheinlich nicht um zwei identische Publikationen. In diesem Fall sollte die Bewertung eines fehlenden Feldes höher eingestuft werden.

In bibliographischen Daten gibt es oftmals bestimmte Felder, die die Daten voneinander „trennen“, wie z.B. der Typ der Publikation oder das Publikationsjahr. Desweiteren sind teils in anderen Feldern Zusatzinformationen gespeichert, die bei der Bestimmung des Abstandes verwendet werden können. Fehlt das Publikationsjahr, so kann dies oftmals aus dem Feld BOOKTITLE extrahiert werden, da bei Konferenzen im BOOKTITLE das Jahr angegeben wird. Als weiteres Kriterium kann die „Zuverlässigkeit“ der Feldinhalte bei der Berechnung der Ähnlichkeit berücksichtigt werden. Bestimmte Felder beinhalten Daten, in denen Tippfehler unwahrscheinlich oder sogar ausgeschlossen sind, weil diese bei der Erfassung mit vordefinierten Wertelisten abgeglichen werden. Unterschiedliche Inhalte in diesen Feldern sollen sich stärker auf den Abstand auswirken als in anderen.

In Abb. 3.2 sieht man, dass es sich beim ersten Datensatz um einen Artikel und im zweiten um einen Konferenzbeitrag handelt. Dieser Unterschied ist ein Indiz dafür, dass es sich um zwei unterschiedliche Daten handelt. Darüber hinaus steht im Feld BOOKTITLE eine Jahreszahl. Diese kann extrahiert und im Feld YEAR eingefügt werden.

Der Map-Komparator verwendet die folgenden Heuristiken, um die Ähnlichkeit zweier Datensätze a und b zu bestimmen. Dabei wird für jeden einzelnen Feldinhalt ein Score berechnet, der angibt, wie ähnlich sich die Feldinhalte der beiden zu vergleichenden Datensätze sind:

Preprocessing: Felder BOOKTITLE und YEAR Falls der Datensatz kein Feld YEAR besitzt, wird aus dem Feld BOOKTITLE die Jahreszahl extrahiert und in das Feld YEAR eingetragen.

Feld YEAR Das Feld YEAR wird nur auf Gleichheit überprüft. Bei unterschiedlichen Jahresangaben werden die Feldinhalte als ungleich betrachtet.

Feld PAGES Im Feld PAGES treten folgende Fehlerquellen auf: Zum einen sind nicht immer die Seitenbereiche angegeben, sondern nur die Startseite, zum anderen stimmen in einigen Fällen die Endeseiten nicht überein. Um diese Fehler zu berücksichtigen wird beim Vergleich der Seitenzahlen wie folgt vorgegangen: Bei identischen Seitenbereichen ist das Ergebnis klar. Aber wie werden unterschiedliche Angaben behandelt? Dazu trennt man die beiden Seitenangaben p_1 und p_2 in Start- und Endeseiten (s_1, e_1) und (s_2, e_2). Ist eine Seitenangabe nur durch die Startseite s_i gegeben, die andere durch Start- und Endeseite (s_j, e_j) und stimmen die Startseiten überein $s_i = s_j$, so handelt es sich um zwei gleiche Seitenbereiche. Sind die Startseiten gleich $s_i = s_j$ und

die Endeseiten der beiden Angaben stimmen nicht überein $e_i \neq e_j$, so definieren sie vielleicht die gleichen Seitenbereiche und es wird ein Score von 0.5 zurückgegeben. In den anderen Fällen sind durch die Seitenangaben unterschiedliche Bereiche definiert.

$$\text{Score}_{\text{PAGES}}((s_1, e_1), (s_2, e_2)) = \begin{cases} 1 & \text{wenn } s_1 = s_2 \wedge (e_1 = e_2 \vee e_1 = \emptyset \vee e_2 = \emptyset) \\ 0,5 & \text{wenn } s_1 = s_2 \wedge e_1 \neq e_2 \\ 0 & \text{wenn } s_1 \neq s_2 \end{cases}$$

Felder AUTHOR **und** EDITOR Wegen der unterschiedlichen Angaben der Namen (Vornamen bzw. Initialen, Akzentzeichen und Namenszusätze (III oder Jr.)) ist ein Stringvergleich auf diesen „Rohdaten" nicht sinnvoll. Um hier diese unterschiedlichen Angaben zu vereinheitlichen, werden zuvor die Autor/-Editornamen durch eine einheitliche Schreibweise ersetzt, wobei jeder Autor/Editor nur durch die Initialen der Vornamen gefolgt von seinem Nachnamen (ohne Jr. oder andere Zusätze) dargestellt wird. Erst auf Basis dieser Normalisierung der Namen erfolgt die Berechnung der Ähnlichkeit.

Felder JOURNAL **oder** SERIES Die Erfassung der Daten dieser Feldinhalte erfolgt mittels einer vordefinierten Werteliste. Ein Tippfehler in diesen Inhalten ist aus diesem Grund ausgeschlossen und es kann deshalb auf eine Berechnung der Edit-Distanz verzichtet werden. Stimmen diese Inhalte überein, erhalten sie einen Score von 1.0, sonst 0.0.

Feld PUBLISHER Auch dieses Feld wird mittels einer vordefinierten Werteliste erfasst, aber ein exakter Stringvergleich liefert in diesem Fall nicht immer das gewünschte Ergebnis. Das Problem liegt darin, dass nicht nur die Verlagsnamen, sondern auch die Verlagsorte angegeben sind und die Verlagsorte oftmals nicht immer komplett oder in der gleichen Reihenfolge angegeben sind. Deshalb werden vor einem Vergleich die Verlagsnamen aus den Inhalten extrahiert und nur diese verarbeitet.

Typ des Datensatzes Der Typ (*artilce, inproceedings, misc ...*) wird nicht mittels eines Stringvergleichs behandelt. Bestimmte Publikationstypen werden als identisch betrachtet, wie z.B. *inproceedings* und *proceedings*, andere dienen als Universaltypen (diese sind mit jedem Typen identisch) (*misc* oder *unpublished*).

Vergleich mit leerem Feldinhalt Ist

$$keys_A \subset keys_B \vee keys_B \subset keys_A$$

gegeben, so wird dieses Feld in der Berechnung der Ähnlichkeit nicht berücksichtigt.

Gewichtung $c(k)$ Die einzelnen Felder erhalten eine Gewichtung. Diese Gewichtung wird bei der Abstandsbestimmung mit berücksichtigt.

Der Abstand $d(A, B)$ wird mit unten stehender Formel und obigen Heuristiken bestimmt:

$$S = keys(A) \cup keys(B)$$

$$d(A,B) = \frac{\sum_{k \in S} \text{Score}_k(value_A(k), value_B(k)) * c(k)}{\sum_{k \in S} c(k)} \tag{3.5}$$

Algorithmus 3.3: Map-Komparator

MapComparator(A, B)

if $empty(value_A(\text{YEAR}))$ **then** $put_A(\text{YEAR}, extractYear(value_A(\text{BOOKTITLE}))$
if $empty(value_B(\text{YEAR}))$ **then** $put_B(\text{YEAR}, extractYear(value_B(\text{BOOKTITLE}))$

$d = 0$
$S = keys(A) \cup keys(B)$
$I = \emptyset$

foreach k **in** S **do**
  if $k = \text{TYPE}$ **then**
    $d = d + calculateType(A,B) * c(k)$
  else if $k = (\text{AUTHOR} \vee \text{EDITOR})$ **then**
    $d = d + calculateAuthor(A,B) * c(k)$
  else if $k = \text{YEAR}$ **then**
    $d = d + calculateYear(A,B) * c(k)$
  else if $k = \text{PAGES}$ **then**
    $d = d + calculatePages(A,B) * c(k)$
  else if $k = (\text{JOURNAL} \vee \text{SERIES})$ **then**
    $d = d + calculateJournalOrSeries(A,B) * c(k)$
  else if $k = (\text{INSTUITUTION} \vee \text{ORGANIZATION})$ **then**
    $d = d + calculateInstututionOrOrganization(A,B) * c(k)$
  else if $k = \text{PUBLISHER}$ **then**
    $d = d + calculatePublisher(A,B) * c(k)$
  else if $(empty(value_A(k)) \vee empty(value_B(k)))$ **then**
    if isMoreDefined(A, B) **then**
      $I = I \cup \{k\}$
    else
      $d = d + 0.25 * c(k)$
  else
    $d = d + dist(value_A(k), value_B(k)) * c(k)$
  fi
od

$d = \frac{d}{\sum_{k \in S \wedge k \notin I} c(k)}$

return d

3.3.7 Vergleich der Verfahren

Welche Konstellationen beim Vergleich zweier Datensätze A und B auftreten können, wird im Folgenden gezeigt:

Fall 1 $keys(A) = keys(B)$**:** Die Schlüssel in beiden Datensätzen sind identisch.

> **Fall 1.1** $A = B$**:** $value_A(k) = value_B(k) : \quad \forall k \in keys(A)$
> Beide Datensätze stimmen in allen Feldern überein.
> $\Rightarrow d(A,B) = 1$

Fall 1.2 $A \neq B$**:** $value_A(k) \neq value_B(k): \quad \forall k \in keys(A)$
Beide Datensätze stimmen in keinem Feld überein.
$\Rightarrow d(A, B) = 0$

Fall 1.3 $A \neq B$**:** $0 < |value_A(k) = value_B(k)| < |keys(A)|: \quad \forall k \in keys(A)$
Beide Datensätze stimmen in einigen Feldern überein.
$\Rightarrow 0 < d(A, B) < 1$

Fall 2 $keys(A) \cap keys(B) = \emptyset$**:** Die Schlüssel in beiden Datensätzen sind verschieden.
Die Datensätze A und B sind nicht miteinander vergleichbar.
$\Rightarrow d(A, B) = 0$

Fall 3 $keys(A) \cap keys(B) = S \neq \emptyset$**:** Die Schlüssel in beiden Datensätzen sind teilweise identisch.

Fall 3.1 $A = B$**:** $value_A(k) = value_B(k): \quad \forall k \in S$
Die Werte stimmen in den gemeinsamen Feldern überein.
$\Rightarrow d(A, B) \approx 1$

Fall 3.2 $A \neq B$**:** $0 < |value_A(k) = value_B(k)| < |S|: \quad \forall k \in S$
Die Werte stimmen in den gemeinsamen Feldern teilweise überein.
$\Rightarrow 0 < d(A, B) < 1$

		Abstand				
Vergleich		1 – 2	1 – 3	2 – 3	1 – 4	1 – 5
Verfahren	Verbesserung 1 ∩	1.00	1.00	1.00	0.00	0.00
	Verbesserung 1 ∪	0.22	0.71	0.29	0.00	0.00
	Verbesserung 2 ∩	1.00	1.00	1.00	0.28	0.19
	Verbesserung 2 ∪	0.22	0.71	0.29	0.21	0.13
	Map-Komparator	0.37	1.00	0.40	0.24	0.18

Tabelle 3.4: Vergleich der Komparatoren

In Tabelle 3.4 sind die Ergebnisse der Berechnung der Abstände zwischen den Daten in Abb. 3.2 zu finden. Dabei bedeutet $\cap : S = keys(A) \cap keys(B)$ und $\cup : S = keys(A) \cup keys(B)$. Die beiden Datensätze 1 und 3 sind Dubletten, alle anderen Datensätze sind verschieden. Wie in der Tabelle ersichtlich ist, erkennen die beiden Verfahren mit $\cap$ nicht die unterschiedlichen Daten 1 und 2. Die beiden Daten 1 und 3 werden richtig als Dubletten erkannt. Die beiden Verfahren mit $\cup$ können zwar die beiden Datensätze 1 und 3 von 2 trennen, allerdings nur mit einem Score von 0.21 bzw. 0.29. Die beiden gleichen Datensätze 1 und 3 werden allerdings nur mit einem Score von 0.71 bewertet. Der Map-Komparator schafft es, ähnliche Daten einem hohen Score zuzuordnen. Da 1 und 2 nicht identisch sind, wird bei diesem Vergleich ein Score von 0.37 bestimmt, für die beiden Daten 1 und 3 ein Score von

1.00. Völlig unterschiedliche Daten erhalten beim Vergleich einen niedrigen, ähnliche Daten hingegen einen sehr hohen Score. Damit ist es möglich mit dem Map-Komparator sehr genau zu unterscheiden, ob es sich um vollständig unterschiedliche, ähnliche oder identische Daten handelt.

Auswirkungen eines Fehlers

Der Map-Komparator erkennt sehr zuverlässig, ob zwei Datensätze identisch oder ähnlich sind oder ob es sich um zwei völlig verschiedene Daten handelt. Das Problem besteht darin, geeignete Schranken γ_1 und γ_2 zu finden, für die gilt: alle Dubletten, deren Score $> \gamma_2$ ist Ⓒ, werden als identisch betrachtet und können automatisch entfernt werden. Alle Dubletten, deren Score $> \gamma_1$ und $\leq \gamma_2$ Ⓑ sind potentielle Dubletten und müssen von Hand überprüft und evtl. entfernt werden. Die Überprüfung von Hand bedeutet einen hohen zeiltichen Aufwand, deshalb sollten die beiden Schranken γ_1 und γ_2 nicht zu weit auseinander liegen. Ist die Schranke γ_2 zu niedrig gewählt, besteht die Gefahr, Daten zu löschen, die keine Dubletten sind. Wird hingegen γ_1 zu hoch gewählt, so gehen potentielle Dubletten verloren und die Datenbasis wird nicht komplett bereinigt.

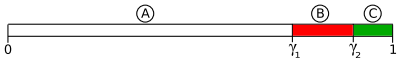


Abbildung 3.6: Schranken γ_1 und γ_2

Für die Daten der LEABib bestimmt der Map-Komparator 280 Dubletten (mit einem Score .0.9), wobei 190 Dubletten einen Score = 1 erhalten. Diese 190 Datensätze sind echte Dubletten, die sofort entfernt werden können. Weitere 74 Dubletten erhielten einen Score, der ≥ 0.95 und < 1 ist. Unter diesen 74 Dubletten befinden sich auch Datensätze, die keine Dubletten sind.

Aber wie wirkt sich ein Fehler in einem Datensatz aus? Dazu betrachten wir folgenden Datensatz:

```
@Article{Craigen/03,
  AUTHOR = {Craigen, R.},
  TITLE = {Boolean and ternary complementary pairs},
  JOURNAL = {J. Comb.~Theory Series A},
  VOLUME = {104},
  NUMBER = {3},
  PAGES = {1-16},
  YEAR = {2003},
}
```

Entfernt man ein oder mehrere Felder aus dem Datensatz, so ermittelt der Map-Komparator immer noch einen Score von 1, da obiger Datensatz mehr definiert ist und somit die fehlenden Felder nicht berücksichtigt werden.

Ändern wir in diesem Datensatz im Titel zwei Buchstaben, so ist der Edit-Abstand der beiden Titel 2 und als Score erhält man: $\frac{0.8*1+0.8*\left(1-\frac{2}{39}\right)+1*1+0.8*1+0.8*1+1*1+1*1}{0.8+0.8+1+0.8+0.8+1+1} \approx 0.993$.
Ein Fehler im Feld AUTHOR wirkt sich nur aus, wenn entweder der Fehler im Nachnamen auftritt, oder sich auf die Initialen der Vornamen auswirkt. Bei einem Fehler ist der ermittelte Score in der Größenordnung von 0.99.
Ändert man den Datensatz folgendermaßen: Entferne das Journal und füge statt dessen einen Serientitel ein. In diesem Fall sinkt der Score auf 0.722, da zwei Felder unterschiedliche Werte aufweisen und diese Felder hoch gewichtet werden.
Ein Fehler in einem Feld, das nur auf Gleichheit überprüft wird, verringert den Score auf 0.86. Ein Fehler in den Seitenzahlen wirkt sich wie folgt aus: unterschiedliche Seitenzahlen - 0.86, gleiche Startseiten - 0.93.

Anhand dieser Daten können die Werte der Schranken für γ_1 und γ_2 festgelegt werden: $\gamma_1 \approx 0.9$ und $\gamma_2 \approx 0.99$. Mit diesen Schranken wird sichergestellt, dass nur echte Dubletten automatisch aus der LEABib gelöscht werden und die Anzahl der potentiellen Dubletten, die von Hand geprüft werden muss, möglichst gering ist. Mit einem geeigneten Tool, mit dem man die Liste der Dubletten abarbeiten kann und das die Unterschiede in beiden potentiellen Dubletten grafisch darstellt ist der Aufwand zur manuellen Nachbearbeitung der Dublettenliste gering.

3.3.8 Probleme

Vergleicht man zwei Elemente a und b mittels einer der obigen Verfahren, so treten folgende Probleme auf:

- **a und b sind äquivalent**:
  In diesem Fall muss die Äquivalenz-Relation *true* liefern. Ist das Ergebnis des Vergleichs allerdings $false$, so werden die beiden äquivalenten Elemente als verschieden interpretiert und man spricht von einem *false-negativ*.

- **a und b sind verschieden**:
  In diesem Fall muss die Äquivalenz-Relation $false$ liefern. Ist das Ergebnis des Vergleichs *true*, so werden die beiden unterschiedlichen Elemente als äquivalent interpretiert und man spricht von einem *false-positiv*.

Die Schwierigkeit besteht darin, eine Äquivalenz-Relation zu finden, die

- die Anzahl der *false-negativ* und *false-positiv* möglichst gering hält, und
- schnell berechnet werden kann.

Durch die Verwendung des Map-Komparators und geeigneten Schranken γ_1 und γ_2 können automatisch gleiche und ähnliche Datensätze identifiziert werden. Alle Daten, deren Score $d(A, B) \geq \gamma_1$ ist, können als Dubletten markiert werden. Alle anderen Daten sind als verschieden zu betrachten. Mit entsprechenden Strategien ist es möglich, alle Dubletten in

einer Menge von Daten mit möglichst wenigen Vergleichen zu bestimmen. Wie die Anzahl der Vergleiche gering gehalten und damit das Finden der Dubletten schnell durchgeführt werden kann, zeigt das nächste Kapitel.

Kapitel 4

Dublettenerkennung

Die Erfassung von Daten und die Pflege von großen, ständig wachsenden Datenbeständen erfordert viel Zeitaufwand. Zum einen ist die Erfassung neuer Daten in hoher Qualität mit großem Aufwand verbunden, zum anderen stellt die Pflege der bestehenden Datenbank (Fehlerbereinigung, Normalisierung der Daten, Dublettenerkennung) eine Herausforderung an die verwendeten Tools.
Dieses Kapitel behandelt die Erkennung von Dubletten in großen Datenbeständen. Dubletten werden mit unterschiedlichen Verfahren erkannt (Ähnlichkeitsmaße, Äquivalenz-Relationen), wobei diese Verfahren immer nur entscheiden können, ob zwei Datensätze identisch sind oder nicht. Da ein Vergleich zweier Datensätze mit hohem Aufwand verbunden ist (siehe Kapitel 3), ist es wichtig, die Anzahl der nötigen Vergleiche zu minimieren.
In diesem Kapitel werden Algorithmen vorgestellt, die in einer bestehenden Datenmenge die Dubletten mit möglichst wenigen Vergleichen zuverlässig erkennen. Diese Algorithmen lassen sich in zwei Kategorien unterteilen: sortierende und nicht sortierende. Zu den sortierenden Algorithmen zählen z.B. das Sliding Window Verfahren [HS95] und [HS98], zu den nicht sortierenden die vollständige Suche, Mergededup oder das Cluster-Verfahren [Ber02].
In vielen Anwendungsfällen ist eine Sortierung der Daten möglich, aber es existieren Datensammlungen, in denen eine Sortierung nicht möglich ist oder zumindest keinen sinnvollen Beitrag bei der Suche nach Dubletten leistet (z.B. Bilder). Trotz einer minimalen Anzahl von Vergleichen soll gewährleistet werden, dass alle Dubletten bzgl. der Äquivalenz-Relation gefunden werden.
Zum Abschluss wird das Problem behandelt, wie mit den gefundenen Dubletten verfahren werden soll. Dubletten einfach zu löschen ist in vielen Fällen nicht die geeignetste Methode.

Definition 4.1:
Sei $S = \{s_1, ..., s_n\}$ eine Menge von Objekten und $f(a, b)$ eine Äquivalenz-Relation auf den Elementen von S. Die Menge $T = \{t_1, ..., t_m\}$ repräsentiert die Äquivalenz-Klassen von S und $r_1 = |t_1|$, ..., $r_m = |t_m|$. O.B.d.A. sei $r_1 \leq r_2 \leq ... \leq r_m$.

Des weiteren sei:

- $n = \sum_{i=1}^{m} r_i$

- $\overline{r} = \frac{\sum_{i=1}^{m} r_i}{m} = \frac{n}{m}$ bezeichnet die durchschnittliche Größe der Äquivalenz-Klassen.
- $\alpha = \frac{m}{n}$ bestimmt die Qualität („Reinheitsgrad“) der Daten.
- $\beta = 1 - \alpha = \frac{n-m}{n}$ bezeichnet den „Verschmutzungsgrad“ der Daten.

$R(m, n)$ bezeichnet die Anzahl der nötigen Vergleiche, um aus n Elementen m Repräsentanten der m Äquivalenz-Klassen zu bestimmen.

4.1 Schranken

Lemma 1 (Schranken für r_m)

$$1 \leq r_m \leq n - m + 1$$

Beweis: Wenn $r_1, ..., r_{m-1} = 1$, dann ist r_m maximal. $\Rightarrow r_m = n - (m - 1) = n - m + 1$

□

Lemma 2 (Schranken für $R(m, n)$) *Es gilt:*

1. $R(1, n) = n - 1$
2. $R(n, n) = \binom{n}{2}$
3. $R(m, n) \leq O(m * n)$

Beweis: Zu 1: Besteht eine Menge S nur aus einer Äquivalenz-Klasse, so reichen $n - 1$ Vergleiche.

Zu 2: Besteht eine Menge S aus n Äquivalenz-Klassen, so müssen alle Elemente miteinander verglichen werden.

$$(n-1) + (n-2) + ... + (1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \binom{n}{2} \approx \frac{n^2}{2}$$

Zu 3: Betrachte folgenden Algorithmus:
Sei L die Liste der Elemente von S.

$x_1 = \text{pop}(L)$
while (L not empty) **do**
    while (L has next) **do**
        $x_2 = \text{next}(L)$
        if ($x_1 \equiv x_2$) **then**

```
            delete x_2 in L
        fi
    od
    x_1 = pop(L)
od
```

Obiger Algorithmus entfernt doppelte Einträge aus der Liste L. Zum Schluss enthält die Liste L nur noch Repräsentanten der Äquivalenz-Klassen aus der Menge S. Im 1. Schritt werden $n-1$ Vergleiche benötigt, im 2. Schritt $n-(r_1+1)$ Vergleiche, ..., im letzten Schritt $n-(r_1+r_2+...+r_m+1)$.

$$
\begin{aligned}
& n-1+n-1-(r_1)+n-1-(r_1+r_2)+\ldots+n-1-(r_1+r_2+\ldots+r_m) \\
= \;& (m+1)(n-1)-r_1-(r_1+r_2)-\ldots-(r_1+r_2+\ldots+r_m) \\
= \;& (m+1)(n-1)-(r_1+(r_1+r_2)+\ldots+(r_1+r_2+\ldots+r_m))
\end{aligned}
$$

Fall 1:

$$
\begin{aligned}
& r_1=r_2=\ldots=r_{m-1}=1 \wedge r_m=(n-m+1) \quad \text{(Worst case)} \\
= \;& (m+1)(n-1)-(1+2+3+\ldots+m-1+n) \\
= \;& (m+1)(n-1)-\left(n+\sum_{i=1}^{m-1} i\right) \\
= \;& (m+1)(n-1)-\frac{2n+m(m-1)}{2} \\
= \;& O(mn)
\end{aligned}
$$

Fall 2:

$$
\begin{aligned}
& r_1=(n-m+1) \wedge r_2=\ldots=r_m=1 \quad \text{(Best case)} \\
= \;& (m+1)(n-1)-((n-m+1)+(n-m+1)+1+ \\
& (n-m+1)+2+\ldots+(n-m+1)+(m-2)+(n-m+1)+(m-1)) \\
= \;& (m+1)(n-1)-\left(m(n-m+1)+\sum_{i=1}^{m-1} i\right) \\
= \;& (m+1)(n-1)-\frac{2m(n-m+1)+m(m-1)}{2} \\
= \;& O(mn)
\end{aligned}
$$

Fall 3:

$$
\begin{aligned}
& \bar{r_i}=\frac{n}{m} \quad \text{(Average case)} \\
= \;& (m+1)(n-1)-\frac{n}{m}-(\frac{n}{m}+\frac{n}{m})-\ldots-(\frac{n}{m}+\frac{n}{m}+\ldots+\frac{n}{m}) \\
= \;& (m+1)(n-1)-\frac{n}{m}\sum_{i=1}^{m} i
\end{aligned}
$$

$$
\begin{aligned}
&= (m+1)(n-1) - \frac{n}{m}\frac{(m+1)m}{2} \\
&= (m+1)(n-1) - \frac{(m+1)n}{2} \\
&= \frac{2(m+1)(n-1) - (m+1)n}{2} \\
&= O(mn)
\end{aligned}
$$

□

4.2 Vollständige Suche

Bei der vollständigen Suche werden alle Elemente der Menge S paarweise miteinander verglichen, d.h. in einer Liste mit n Elementen wird jedes Element e mit $n-1$ anderen Elementen aus S verglichen. Auf Grund der Symmetrie der Äquivalenz-Relation kann die Anzahl der Vergleiche halbiert werden, da nach einem Vergleich $a =^? b$ der Vergleich $b =^? a$ nicht mehr nötig ist.

Algorithmus 4.1: Vollständige Suche

for $i_1 = 0$ **to** n **do**
    for $(i_2 = i_1 + 1$ **to** $n) \wedge (i_1 \neq i_2)$ **do**
        if $(s_{i_1} \equiv s_{i_2})$ **then**
            mark x_2 as Duplicate
        fi
    od
od

Analyse

Die Anzahl der Vergleiche beträgt $O(n^2)$.

Beweis: Der erste Datensatz wird mit $n-1$ anderen verglichen, der 2. mit $n-2$, der vorletzte mit 1.

$$
\begin{aligned}
(n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} i &= \frac{(n-1)n}{2} \\
&= O(n^2)
\end{aligned}
$$

□

Die vollständige Suche findet alle Dubletten in einer Datenmenge. Da allerdings $O(n^2)$ Vergleiche zwischen den einzelnen Daten nötig sind, ist dieses Verfahren nur für kleine Datenmengen anwendbar.

4.3 Mergededup

Mergededup (siehe Abb. 4.1) führt eine vollständige Suche durch und basiert auf der Idee von Mergesort. Die Menge S wird rekursiv in jeweils 2 Teilmengen S_1 und S_2 zerlegt ($|S_1| = |S_2|$), bis die Teilmengen einelementig sind. Anschließend werden die einzelnen Listen wieder zusammengefasst. Dabei werden die Elemente aus S_1 und S_2 miteinander verglichen. Sind zwei Elemente $v \in S_1$ und $w \in S_2$ „äquivalent“, so wird $w \in S_2$ entfernt.

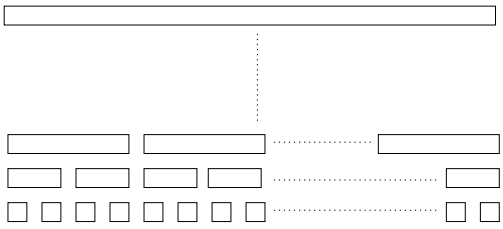


Abbildung 4.1: Funktionsweise von Mergededup

Beobachtung

- Die Höhe h des Baumes ist auf $\log(n)$ beschränkt.
- In Tiefe t gibt es 2^t Listen.
- In Tiefe t enthalten die Listen jeweils $\frac{n}{2^t}$ Elemente.
- Um zwei Listen L_1, L_2 mit $k_1 = |L_1|$ und $k_2 = |L_2|$ Elementen zu vergleichen, sind $k_1 * k_2$ Vergleiche nötig.

Lemma 3 *Um die Listen in Tiefe t zu vergleichen, sind*

$$\frac{1}{2} \cdot 2^t \left(\frac{n}{2^t}\right)^2 = \frac{n^2}{2^{t+1}}$$

Vergleiche nötig.

Analyse

Lemma 4 *Mergededup benötigt maximal $\frac{n(n-1)}{2}$ Vergleiche.*

Beweis: Der Merge-Baum hat Höhe $\log n$.

$$\sum_{t=1}^{\log n} \frac{n^2}{2^{t+1}} = n^2 \left(\sum_{t=1}^{\log n} \left(\frac{1}{2}\right)^{t+1} \right) = n^2 \sum_{t=0}^{\log n} \left(\frac{1}{2}\right)^{t+1} - \frac{1}{2}^1$$

$$= n^2 \sum_{t=0}^{\log n} \left(\frac{1}{2}\right)^{t} - \frac{1}{2}^0 - \frac{1}{2}^1 + \frac{1}{2}^{\log(n)+1} = n^2 \left(\frac{\left(\frac{1}{2}\right)^{\log(n)+1} - 1}{\frac{1}{2} - 1} - \frac{3}{2} + \left(\frac{1}{2}\right)^{\log(n)+1} \right)$$

$$= n^2 \left(-2 * \left(\frac{1}{2}\right)^{\log(n)+1} + 2 - \frac{3}{2} + \left(\frac{1}{2}\right)^{\log(n)+1} \right) = n^2 \left(\frac{1}{2} - \left(\frac{1}{2}\right)^{\log(n)+1} \right)$$

$$= n^2 \left(\frac{1}{2} - \frac{1}{2n} \right) = \frac{n(n-1)}{2}$$

□

Obwohl Mergededup im Worst case eine Laufzeit von $O(n^2)$ besitzt, zeigt der Algorithmus in realen Umgebungen eine Laufzeit von $O(n*m)$. Abb. 4.2 zeigt die Anzahl der benötigten Vergleiche bei einem Verschmutzungsgrad von 20% in Abhängigkeit der Eingabegröße n.

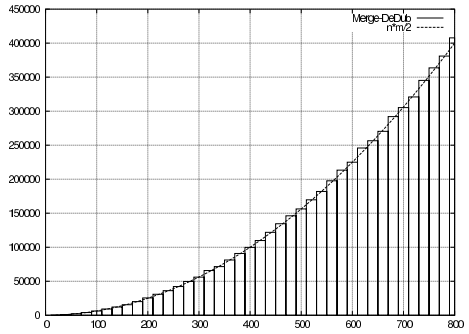


Abbildung 4.2: Anzahl der Vergleiche für $q = 0.8$

4.4 Clustering

Die vollständige Suche nach Dubletten in einer Menge S betrachtet alle Paare von Daten. Viele dieser Vergleiche liefern als Ergebnis, dass die beiden betrachteten Datensätze ungleich sind, da sie sehr verschieden sind. Beim Clustering nutzt man diese Eigenschaft der „Verschiedenheit“ der Daten aus. Eine Funktion $f(x) : S \rightarrow \text{STRING}$ weist jedem Datensatz in S einen Schlüssel zu. Der Schlüssel k_i besteht aus bestimmten Feldinhalten, oder Teilen daraus. Alle Datensätze e_i mit gleichem Schlüssel k_i werden in einem Cluster zusammengefasst.

Findet man eine Funktion f, die die Schlüssel so erzeugt, dass die Cluster genau die Äquivalenz-Klassen bilden, ist keine weitere Suche nach Dubletten mehr nötig. In den meisten Fällen wird allerdings eine solche Funktion f nicht existieren oder nur sehr schwer zu bestimmen sein. In diesem Fall bilden die Cluster keine Äquivalenz-Klassen, sondern dienen als Mengen C_j von Elementen e_{j_i}, die „ähnliche Elemente“ zusammenfassen.

Beim Clustering wendet man einen Algorithmus zur Dublettenerkennung nur jeweils auf den Daten innerhalb eines Clusters C_j an. Wichtig ist die Wahl einer geeigneten Funktion f, die die Cluster bildet. Werden zu viele Elemente in einem Cluster zusammengefasst ($|C_j| \approx O(n)$), so liegt die Laufzeit wiederum bei der Laufzeit des Algorithmus zur Dublettenerkennung, andernfalls beträgt die Laufzeit

$$k * LZ_{Alg.Dublettenerkennung}(\frac{n}{k})$$

Hierbei gibt k die Anzahl der Cluster an und die Funktion f unterteilt die Menge S in etwa gleich große Cluster C_j ($\forall i, j : |C_i| \approx |C_j|$). Bei Verwendung der vollständigen Suche innerhalb der einzelnen Cluster ergibt sich eine Laufzeit von $k * O\left(\frac{n}{k}\right)^2 = O\left(\frac{kn^2}{k^2}\right) = O\left(\frac{n^2}{k}\right)$.

Zerlegt die Funktion f die Menge S in zu viele kleine Cluster, so werden die Dubletten nicht vollständig erkannt, da viele Dubletten in unterschiedliche Cluster eingefügt werden. Abhilfe schafft hier ein Multipass-Verfahren. Hierbei wird der Algorithmus mehrmals ausgeführt, wobei in jedem Durchlauf eine andere Funktion f zum Erzeugen der Schlüssel verwendet wird und somit auch die Cluster in jedem Durchgang aus verschiedenen Elementen bestehen.

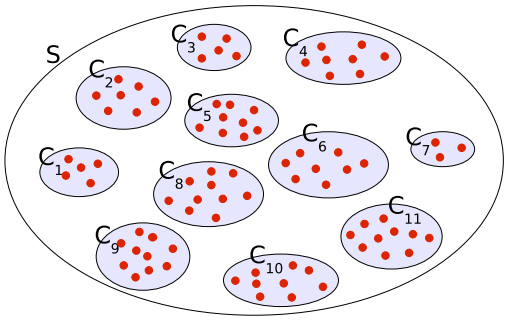


Abbildung 4.3: Clustering

Algorithmus 4.2: Clustering

Generate keys with f
Make Cluster $C_1, \ldots, C_i$

foreach C_i **do**
    Dublettenerkennung in C_i
od

Beispiel:
Das Portal io-port.net verwendet ein Cluster-Verfahren, um Dubletten zu erkennen. Dabei wird folgende Methode zum Erzeugen der Schlüssel für die einzelnen Datensätze verwendet. Aus folgenden Feldinhalten wird ein STRING aufgebaut:

- der TYPE
- der TITEL,
- das YEAR,
- von allen Autoren (AUTHOR) die Nachnamen
- falls TYPE = „article“, das VOLUME

Dieser so erzeugte String wird normalisiert (d.h. alle Zeichen außer a-z, A-Z und 0-9 werden entfernt) und seine MD5-Prüfsumme gebildet. Diese MD5-Prüfsumme ist der Schlüssel des Datensatzes. Alle Datensätze mit gleichem Schlüssel werden als Dubletten betrachtet. Es erfolgt keine weitere Suche nach Dubletten innerhalb eines Clusters. Das Bilden der MD5-Prüfsumme ist für das Verfahren nicht nötig, da durch die Prüfsumme die Cluster nicht verändert werden.
Dieses Vorgehen findet einen Großteil der Dubletten, aber es bleiben jedoch eine Vielzahl offensichtlicher Dubletten unerkannt. Da der Schlüssel eines jeden Datensatzes auch Felder enthält, in denen häufig unterschiedliche Schreibweisen (z.B. Formeln) auftreten, werden für „gleiche“ Daten unterschiedliche Schlüssel erzeugt.
□

Für die LEABib ist das Verfahren in obigem Beispiel zur Erzeugung der Cluster nicht geeignet. Aber welche Daten sind geeignet und wie kann ein Schlüssel erzeugt werden, der die entsprechenden Cluster bildet und robust gegenüber den typischen Fehlern in den Daten ist?
Die erzeugten Cluster sollen zwei Eigenschaften erfüllen:

- Sie sollten möglichst klein sein und
- die potentiellen Dubletten enthalten.

Die Felder JOURNAL oder SERIES würden sich wegen der einheitlichen Schreibweise auf Grund der vordefinierten Wertelisten eignen, aber die Cluster wären teilweise sehr groß (für LNCS ca 12.000 Daten).
Betrachtet man verschiedene Dubletten, so zeichnen sie sich dadurch aus, dass zumindest die Titel und die Autoren gleich sind. Da in beiden Feldern aber Fehler enthalten sind, ist ein Schlüssel, der aus den kompletten Inhalten besteht, ungeeignet. Aus den Titeln lassen sich kaum Teile extrahieren, die robust gegenüber Fehlern sind. Bei den Autoren können allerdings solche Teile gefunden werden. Nimmt man von jedem Autor den normalisierten Nachnamen, und hiervon auch nur den ersten Buchstaben und fügt diese Zeichen aneinander, so erhält man einen Schlüssel, der zum einen potentielle Dubletten zusammenfasst und robust gegenüber Schreibfehlern ist[1]. Um die Clustergröße weiter zu verkleinern, wird der Schlüssel um ein weiteres Feld ergänzt: das Feld YEAR. Das Problem liegt hierbei darin, dass nicht in allen Daten dieses Feld angegeben ist. Um dieses Problem zu umgehen, sammelt man zuerst alle Daten auf, in denen das Feld YEAR fehlt (diese erhalten einen Schlüssel, der nur aus den Namen der Autoren besteht) und fügt in einem Post-Processing diese Daten in alle Cluster ein, die den gleichen Schlüssel auf Basis der Autoren besitzen.

Beispiel:
Die Datensätze aus Abb. 3.2 und 4.5 erzeugen mit obiger Funktion folgende Cluster:

MS92 = { Mirwald-Schnorr/92, Mirwald-Schnorr/?? }
C03 = { Craigen/03 }
BRSV05 = { Bohler-Reith-Schnoor-Vollmer/05 }
N94 = { Neff/94, Neff/94a }
N90 = { Neff/90 }

□

Dieser Algorithmus erzeugt in der Regel robuste und kleine Cluster, in denen die potentiellen Dubletten zusammengefasst sind. In der LEABib sind ca. 350 Daten ohne Angabe von YEAR. Es werden 30842 Cluster erzeugt, wobei die Größe der Cluster von 1 bis 217 variiert und die durchschnittliche Clustergröße bei 3.06 liegt[2] .

4.5 Sliding Window

Das Sliding Window Verfahren ist eine Variante der vollständigen Suche und setzt voraus, dass zu jedem Datensatz s_i aus der Menge S mit Hilfe einer Abbildung $f : s_i \rightarrow String$ ein Schlüssel erzeugt werden kann und für diesen Schlüssel eine Sortierfunktion existiert.

[1] Es wird hierbei vorausgesetzt, dass Fehler im ersten Buchstaben nicht oder nur sehr selten vorkommen.
[2] 3.06 * 30842 = 94376. Die LEABib enthält allerdings nur 82.300 Daten. Die Differenz von ca. 12.000 Daten stammt von den mehrmals eingefügten Daten ohne YEAR.

Algorithmus 4.3: Clustering(S)

$C = \emptyset$
$E = \emptyset$
forall $x \in S$ **do**
    if empty($value_x$(YEAR)) **then**
        $E = E \cup \{x\}$
        $value_x$(CLUSTERKEY) = $generateKey(value_x$(AUTHOR))
    else
        $key = generateKey(value_x$(AUTHOR), $value_x$(YEAR))
        put x into cluster C_{key}
    fi
od
forall $x \in E$ **do**
    $key = value_x$(CLUSTERKEY)
    put x into all clusters $C_{\text{CLUSTERKEY}}$ where CLUSTERKEY begins with key
od

Die Daten werden zuerst anhand des Schlüssels $f(s)$ sortiert und anschließend miteinander verglichen. Dabei werden aber nicht alle n Daten untereinander verglichen, sondern nur diese, die innerhalb eines vorgegebenen Fensters der Größe m liegen.
Man unterscheidet zwischen 1-pass- und Multipass Sliding Window. Beim Multipass Sliding Window wird das Verfahren mit jeweils verschiedenen Funktionen zur Erzeugung der Schlüssel ausgeführt.

Das Verfahren eignet sich zum Deduplizieren von Daten, für die ein Schlüssel generiert werden kann, auf den man eine Sortierfunktion anwenden kann. Dies ist aber nicht immer möglich, bzw. trivial, wie z.B. bei Bildern.
Ein weiteres Problem stellt die Wahl der Funktion f zum Erzeugen der Schlüssel dar. Der Schlüssel sollte die Eigenschaft besitzen, dass identische oder ähnliche Daten in der Sortierung nahe beieinander liegen. Geeignete Funktionen sind z.B.:

- Konkatenation von Feldinhalten
- Romanisierung der Inhalte
- Phonetische Schreibweise

Durch Kombination der obigen Funktionen lassen sich sehr gute Schlüssel für das Sliding Window Verfahren erzeugen. Ein Problem in der Sortierung zeigt sich bei Daten, die einen Fehler enthalten, der sich im Schlüssel auf den Anfang auswirkt. Hierbei wird der Datensatz nicht in der Nähe der ähnlichen Daten einsortiert, sondern weiter entfernt. Da die Fenstergröße klein gewählt wird, um die Anzahl der Vergleiche gering zu halten, wird der fehlerhafte

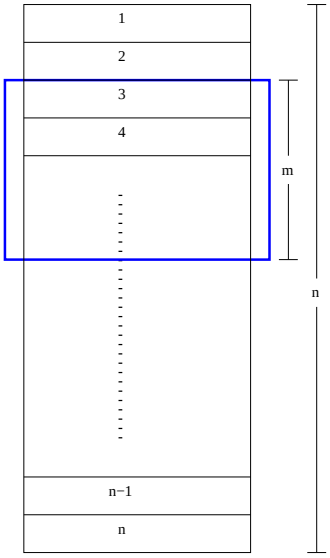


Abbildung 4.4: Sliding Window

Algorithmus 4.4: Sliding Window

Generate keys
Sort S by keys
for $i_1 = 0$ **to** n **do**
    for $(j = i + 1$ **to** $i + m)$ **do**
        if $(x_i \equiv x_j)$ **then**
            mark x_j as Duplicate
        fi
    od
od
for $i_1 = 0$ **to** n **do**
    if (x_i is marked as duplicate) **then**
        remove x_i
    fi
od

Datensatz nicht als Dublette erkannt. Aus diesem Grund wird meist ein Multipass Sliding Window Verfahren angewendet, da sich hier Fehler in den Daten im Allgemeinen nicht in allen Schlüsselfunktionen gleich stark auswirken.
Als unbrauchbar erweisen sich Hashfunktionen, da diese die Eigenschaft besitzen, dass eine kleine Änderung in den Daten eine große Änderung im Schlüssel zur Folge hat. Beim anschließenden Sortieren liegen ähnliche Daten nicht nahe beieinander, sondern weit entfernt. Um diese Daten als Dubletten erkennen zu können, muss die Fenstergröße sehr groß gewählt werden, wobei sich der Geschwindigkeitsvorteil gegenüber der vollständigen Suche verringert, je größer das Fenster wird.

Analyse

Das Sliding Window Verfahren benötigt für die einzelnen Phasen folgende Zeiten:

1. Erzeugen der Schlüssel:	$O(n)$
2. Sortieren der Daten:	$O(n \log n)$
3. Dublettensuche:	$O(m * n)$

Daraus ergibt sich eine Laufzeit von $O(n(\log n + m))$ und $O(n)$ zusätzlichen Speicher, um die Schlüssel zu verwalten.
Das k-pass Sliding Window Verfahren benötigt $O(kn(\log n + m))$ Zeit .

4.6 Nachbarschaften

Die Suche nach Dubletten in einer Menge M mittels der vollständigen Suche benötigt sehr viele Vergleiche und somit auch entsprechend viel Zeit. Dieses Verfahren erkennt dabei alle Dubletten in der Menge S. Das Sliding Window Verfahren hingegen bietet eine bessere Laufzeit, wobei es aber bei ungeschickter Wahl der Funktionen zur Schlüsselerzeugung und entsprechenden Fehlern in den Daten passieren kann, dass nicht alle Dubletten richtig erkannt werden. Wie lässt sich dieses Problem lösen?
Betrachten wir die Datensätze in Abb. 4.5. Bei den ersten beiden Datensätzen handelt es sich um Dubletten. Der 3. Datensatz ist mit den anderen beiden nicht identisch. Bei ungeeigneter Wahl der Funktion zur Schlüsselgenerierung kommen die beiden identischen Datensätze nicht innerhalb eines Fensters zu liegen.
Erstellt man einen Graphen, der die Beziehungen „ist Autor von“ und „wurde veröffentlicht in“ darstellt, so erhält man den in Abb. 4.6 gezeigten Graphen. Anhand des Graphen kann man erkennen, dass die beiden Publikationen *Neff/94* und *Neff/94a* durch das Journal *J. Comput. Syst. Sci.* miteinander verbunden sind. Ebenso sind die beiden Publikationen *Neff/94* und *Neff/90* durch den gemeinsamen Autor *Neff, C. Andrew* verbunden.
Diese zusätzlichen Informationen, die im Publikationsgraphen gespeichert sind, können bei der Suche nach Dubletten verwendet werden.

```
@Article{Neff/94,
  AUTHOR = {Neff, C. Andrew},
  TITLE = {Specified precision polynomial root isolation is in NC},
  JOURNAL = {J. Comput.~Syst.~Sci.},
  VOLUME = {48},
  PAGES = {429-463},
  PUBLISHER = {Academic Press},
  ADDRESS = {New York-San Francisco-London-San Diego},
}

@Article{Neff/94a,
  AUTHOR = {Neff, C.A.},
  TITLE = {specified precision polynomial root isolation is in $NC$},
  JOURNAL = {J. Comput.~Syst.~Sci.},
  VOLUME = {48},
  NUMBER = {3},
  PAGES = {429-463},
  YEAR = {1994},
}

@Techreport{Neff/90,
  AUTHOR = {Neff, C. Andrew},
  TITLE = {Specified precision polynomial root isolation is in $NC$},
  NUMBER = {RC 15653},
  YEAR = {1990, April},
  INSTITUTION = {IBM T.J. Watson Research Center,
                 Yorktown Heights, NY, USA},
}
```

Abbildung 4.5: bibliographische Datensätze

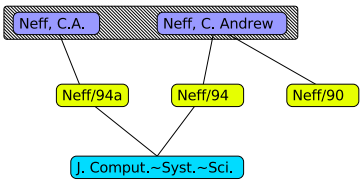


Abbildung 4.6: Publikationsgraph

4.7 Sliding Window*

Dieses Verfahren stellt eine Erweiterung des Sliding Window Verfahrens dar. Beim Sliding Window Verfahren erfolgt die Suche nach Dubletten innerhalb eines fest vorgegebenen Fensters. Probleme stellen Fehler in den Daten dar, die sich so auf die Sortierung auswirken, dass zwei identische Datensätze nicht innerhalb des Vergleichsfensters zu liegen kommen. Hierbei werden die beiden gleichen Daten nicht als Dubletten erkannt und bleiben somit im Datenbestand bestehen. Auch bei einer Vergrößerung des Fensters oder beim Multipass Sliding Window Verfahren bleibt dieses Problem weiterhin bestehen. Es treten immer wieder Fehler auf, so dass zwei gleiche Datensätze in der sortierten Liste nicht innerhalb eines Fensters zu liegen kommen. Um dieses Problem zu umgehen ist es nötig, das zu durchsuchende Fenster zu erweitern, allerdings nicht um eine feste Größe, sondern um Datensätze, die potentielle Dubletten darstellen. Diese potentiellen Dubletten können im Publikationsgraphen gefunden werden. Das Fenster wird um diese zusätzlichen Daten erweitert und der Vergleich erfolgt anschließend innerhalb dieses neuen Fensters.
In einem ersten Schritt wird der Publikationsgraph mit den Beziehungen „ist Autor von“ aufgebaut. Bei den Autorennamen wird jeweils nur der Nachname verwendet, wobei alle Autoren als identisch angesehen werden, die den gleichen Nachnamen besitzen[3]. Somit werden im Graphen 4.6 die beiden Autoren *Neff, C.A.* und *Neff, C. Andrew* als identisch angesehen und mit nur einem Knoten im Graphen repräsentiert. Der grau schraffierte Knoten in 4.6 stellt dieses Zusammenfassen von Knoten dar.
Auf diese Weise entsteht ein Graph, der alle Publikationen mit den Autoren verbindet. Betrachtet man in diesem Graphen eine Publikation p_i, so beschreiben alle Nachbarknoten die einzelnen Autoren a_{i_j} von p_i. Eine zu p_i identische Publikation p_j muss mindestens auch in den Autoren mit der Publikation p_i übereinstimmen. Im Graphen lassen sich diese potentiellen Dubletten leicht bestimmen. Alle Nachbarn der Autoren a_{i_j} stellen die Publikationen dar, die von diesen Autoren verfasst wurden. Hierbei wird nicht nur nach den Publikationen gesucht, die von allen Autoren der Originalpublikation p_i verfasst wurden, sondern alle Publikationen, bei denen mindestens einer der Autoren von p_i Mitautor ist. Diese Vorgehensweise stellt sicher, dass auch Publikationen gefunden werden, in denen evtl. nicht alle Autoren angegeben sind. Oftmals werden bei mehreren Autoren nur die ersten genannt und die weiteren mit z.B. ...ersetzt. Da Autorengruppen über einen bestimmten Zeitraum meist konstant bleiben, ist die Wahrscheinlichkeit hoch, dass zumindest einer der Autorengruppe in den Autoren der Publikation angegeben ist. Die potentiellen Dubletten lassen sich im Publikationsgraphen mittels einer auf Tiefe 2 beschränkten Breitensuche bestimmen.
Im Sliding Window Verfahren wird anschließend nicht nur innerhalb des Fensters der Größe m nach Dubletten gesucht, sondern es wird die Menge der zu durchsuchenden Elemente um die potentiellen Dubletten aus dem Publikationsgraphgen erweitert. Auf diese Weise „vergrößert“ sich automatisch das Fenster.

[3] der Nachname wird zusätzlich normalisiert

Algorithmus 4.5: BFS

$S = \emptyset$

foreach neighbour a_j of p_i **do**
    foreach neighbour p_j of a_j **do**
        $S = S \cup p_j$
    od
od

Algorithmus 4.6: Sliding Window*

Generate Publication-Graph from X
Generate keys $\forall x \in X$
Sort X by keys

for $i_1 = 0$ **to** n **do**
    $S = Subset(X, i, i + m)$
    $S = S \cup BFS(x_i)$
    $x = S_0$
    for $i_2 = 1$ **to** $|S|$ **do**
        if $(x \equiv S_{i_2})$ **then**
            mark S_{i_2} as Duplicate
        fi
    od
od

for $i_1 = 0$ **to** n **do**
    if (x_i is marked as duplicate) **then**
        remove x_i
    fi
od

Analyse

Die LEABib enthält zur Zeit ca. 82.300 Publikationen. Diese wurden von 166658 Autoren verfasst. Das bedeutet, dass im Durchschnitt jeweils 2 Autoren eine Publikation verfassen. Erzeugt man nach obigem Verfahren den Publikationsgraphen für die LEABib, so erhält man 30176 unterschiedliche Knoten für die Autoren.

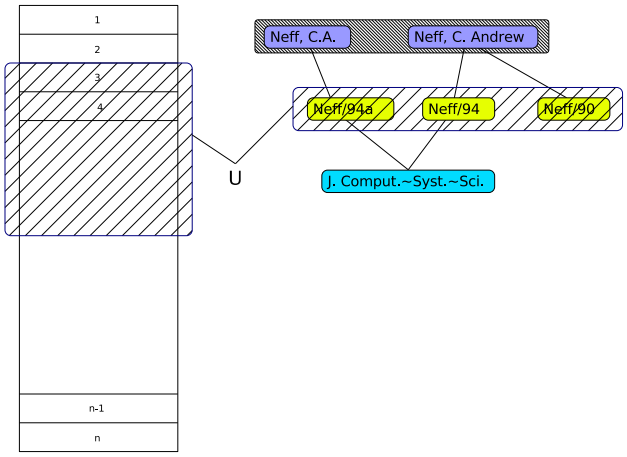


Abbildung 4.7: Sliding Window*

Bei einer vorgegebenen Fenstergröße von 10 ergibt sich eine durchschnittliche Fenstergröße von 117 durch das Hinzufügen der Publikationen der Mitautoren. Die Anzahl der hinzugefügten Publikationen schwankt zwischen 0 und 2585.

Durch dieses Verfahren ist es möglich, trotz einer kleinen Fenstergröße die Dubletten mit möglichst wenigen Vergleichen zu bestimmen. In Bezug auf bibliographische Daten reicht im Allgemeinen ein einziger Durchgang aus, um alle Dubletten zu erkennen. In Kap. 3 wurden verschiedene Verfahren gezeigt, die zwei bibliographische Datensätze vergleichen. Jedes dieser Verfahren liefert einen Score zwischen 0 und 1, der angibt, wie ähnlich sich die beiden zu vergleichenden Daten sind. Der Sliding Window* Algorithmus prüft mittels $\equiv$ die beiden zu vergleichenden Daten auf Gleichheit. Dieses Vorgehen ist durch eines der Vergleichsverfahren aus Kap. 3 zu ersetzen. Da aber diese nicht *true* oder $false$, sondern einen Score zwischen 0 und 1 liefern, ist eine geeignete Schranke γ_1 zu finden, die angibt, ab welchen Score zwei Datensätze als gleich angesehen werden. Das Ergebnis der Deduplizierung ist eine Liste von möglichen Dubletten incl. ihrer Scores.

Falls vorausgesetzt werden kann, dass das Vergleichsverfahren einen Score von 1 nur dann liefert, wenn die beiden Datensätze identisch sind, dann können alle Dubletten mit einem Score von 1 automatisch gelöscht werden. Für alle anderen potentiellen Dubletten ist ein interaktives Vorgehen nötig, um zu vermeiden, dass *false positive*-Daten gelöscht werden. Um dieses interaktive Löschen von Daten zu beschleunigen, ist es möglich, nicht nur die Dubletten mit einem Score von 1 automatisch zu entfernen, sondern alle Daten, deren Score $\geq \gamma_2$ ist. Je niedriger diese Schranke γ_2 gewählt wird, desto geringer ist der Aufwand bei der interaktiven Deduplizierung, aber desto höher ist die Wahrscheinlichkeit, das ein *false positive*-Datensatz aus dem Datenbestand gelöscht wird.

4.8 Vergleich der Verfahren

Testumgebung Um das Sliding Window-, Sliding Window*- und Cluster-Verfahren im Zusammenspiel mit dem Map-Komparator aus Kap. 3.3.6 anhand von realen Daten zu testen, wurde eine Datenbasis mit 17387 Publikationen geschaffen. In dieser Datenbasis existieren keine Dubletten. Aus diesem Datenbestand wurden anschließend 10 beliebige Datensätze ausgewählt und in diese verschiedene typische Fehler eingefügt:

- Fehler im Titel (Rechtschreibfehler, fehlende Formelzeichen)
- fehlende Angaben (Journalname, Publikationsjahr)
- fehlende Akzentzeichen in den Autorennamen
- Vornamen ausgeschrieben oder Vornamen als Initialen

Diese fehlerhaften Daten wurden anschließend als Dubletten nochmals in den Datenbestand eingefügt. Somit enthält die komplette Datenbasis 17397 Datensätze. Auf diesen Datenbestand wurden die Verfahren mit den unterschiedlichen Vergleichsverfahren aus Kapitel 3 getestet. In den Verfahren wurde der Parameter $\gamma_1 = 0.90$ und eine Fenstergröße von 10 verwendet. Beim Sliding Window-Verfahren wurde ein 2-pass Verfahren verwendet. Im 1. Durchgang erfolgte die Sortierung anhand der Titel, im 2. Durchgang anhand des Journal-, Serien- oder Buchtitels, gefolgt vom Volume und der Nummer, und falls das Volume nicht gegeben ist, vom Publikationsjahr. Das Sliding Window*-Verfahren nutzt nur einen Durchgang und verwendet hierbei die Sortierung anhand der Titel. Das Cluster-Verfahren benutzt die in Abschnitt 4.4 angegebene Methode zum Erzeugen der Cluster, die Suche nach Dubletten innerhalb eines Clusters erfolgt mittels einer vollständigen Suche.

Ergebnis Das Sliding Window-Verfahren konnte in keinem Fall alle Dubletten erkennen. Das Problem lag daran, dass die Fehler gezielt so eingetragen wurden, dass in keiner Sortierung die Dubletten innerhalb eines Fensters zu liegen kamen. Das Sliding Window*- und auch das Cluster-Verfahren konnten alle Dubletten erkennen, wobei das Sliding Window*-Verfahren ca. 14-mal so viele Vergleiche benötigte wie das Cluster-Verfahren.

Der Map-Komparator konnte in Verbindung mit dem Sliding Window*-Verfahren alle Dubletten sicher erkennen, ohne dabei auch nur einen *false-positiv* anzugeben. Die Laufzeit betrug dabei allerdings knapp 7 Minuten. Die anderen Kombinationen konnten entweder nicht alle Dubletten finden oder sie hatten mehrere bis viele *false-positiv* in der Liste der gefunden Dubletten. Die schnellste, aber auch unzuverlässigste Methode, der Title-Komparator, benötigte lediglich 8 Sekunden, um die Liste der Dubletten zu bestimmen. Dabei waren 347830 Vergleiche nötig und somit dauert im Schnitt ein Vergleich ca. 1.15 ms.
Das Sliding Window*-Verfahren benötigt ca. die 14-fache Anzahl von Vergleichen und somit auch die 14-fache Zeit. Die Fenstergröße schwankt zwischen 6 und 826. Durchschnittlich

beträgt sie 47. Beim Cluster-Verfahren schwankt die Clustergröße zwischen 1 und 60, wobei 9350 Cluster erzeugt wurden. Die durchschnittliche Clustergröße liegt bei 1.9.
Das Cluster-Verfahren ist in Bezug auf Anzahl der nötigen Vergleiche und somit auch der Laufzeit dem Sliding Window*-Verfahren vorzuziehen, wobei beim Cluster-Verfahren die Anforderungen an die Funktion zum Erzeugen der Cluster nicht immer erfüllt sind (man füge nur eine weitere Dublette ein, in der bei einem Autor der erste Buchstabe falsch ist).
Das Sliding Window*-Verfahren stellt keine zusätzlichen Anforderungen an die Daten. Auch Fehler in den Autorennamen, durch deren Beziehung das Suchfenster vergrößert wird, sind hier zugelassen und führen in den meisten Fällen trotzdem zum richtigen Ergebnis, da es sehr unwahrscheinlich ist, dass in allen Nachnamen der Mitautoren sich Fehler einschleichen.
Ist es nicht möglich, eine geeignete Methode zum Erzeugen der Cluster zu finden, so bietet das Sliding Window*-Verfahren die beste Möglichkeit, alle Dubletten zu finden, allerdings auf Kosten der Laufzeit.
Die folgenden Tabellen zeigen die Anzahl der *false-positiv*, *false-negativ* und der richtig erkannten Datensätze bzgl. des verwendeten Komparators.

Sliding Window

	false-positive	*false-negativ*	richtig	Anzahl Vergleiche
Title-Komparator[4]	19	6	4	347630
Verbesserung 1 ($\cap$)	0	6	4	
Verbesserung 1 ($\cup$)	0	7	3	
Verbesserung 2 ($\cap$)	19	1	9	
Verbesserung 2 ($\cup$)	18	3	7	
Map-Komparator	0	1	9	

Sliding Window*

	false-positive	*false-negativ*	richtig	Anzahl Vergleiche
Title-Komparator[4]	19	5	5	347830
Verbesserung 1 ($\cap$)	0	5	5	
Verbesserung 1 ($\cup$)	0	6	4	
Verbesserung 2 ($\cap$)	19	0	10	
Verbesserung 2 ($\cup$)	18	2	8	
Map-Komparator	0	0	10	

Tabelle 4.1: Vergleich Sliding Window mit Sliding Window*

Wendet man die beiden Verfahren auf die LEABib an, so ergeben sich folgende Werte:

[4] Der Title-Komparator liefert 1, falls die Titel der beiden zu vergleichenden Publikationen identisch sind, andernfalls ist das Ergebnis 0.

	gefundene Dubletten	Fenster/Clustergröße min	max	*avg*	Anzahl Vergleiche	Zeit
Sliding Window*	10	6	826	47	830386	401 s
Clustering	10	1	60	2	57078	27 s

Tabelle 4.2: Vergleich Sliding Window* (Map-Komparator) und Clustering mit Testdaten

	gefundene Dubletten	Fenster/Clustergröße min	max	*avg*	Anzahl Vergleiche	Zeit
Sliding Window*	280	5	2616	118	9736486	7222 s
Clustering	278	1	217	3	1246383	649 s

Tabelle 4.3: Vergleich Sliding Window* (Map-Komparator) und Clustering mit LEABib-Daten

4.9 Zusammenfassen der Daten

Die einzelnen Verfahren zur Dublettenerkennung bestimmen jeweils einen Wert, der angibt, wie ähnlich sich zwei Datensätze u und v sind. Dieser Wert bewegt sich zwischen 0 und 1, wobei 0 vollkommen verschieden und 1 identisch bedeutet. Ab einem Schwellwert γ_2 (ca. 0.99) werden zwei Datensätze als identisch betrachtet. Aber wie sollen die beiden Datensätze behandelt werden? Kann einfach ein beliebiger der beiden gelöscht werden?
Der Map-Komparator liefert einen Score von 1, falls die beiden Datensätze vollkommen identisch sind, oder einer der beiden mehr definiert ist und alle anderen Felder übereinstimmen. Bei einem Score < 1 handelt es sich evtl. um identische Datensätze, wobei zumindest ein Datensatz Fehler enthält.
Die Schwierigkeit besteht darin, durch das Löschen einzelner „Dubletten“ keine Informationen zu verlieren. Versucht man, die Daten des zu löschenden Datensatzes v in den verbleibenden u zu übernehmen, so bleibt die Information, die in v steckt in der Datenbank durch den um v erweiterten Datensatz u erhalten.

Algorithmus 4.7: MergeData

forall $key \in keys_v$ **do**
  if $(value_u(key) == empty)$ **then**
    $value_u(key) = value_v(key)$
  fi
od

Das Ergebnis in Abb. 4.8 zeigt einen zusammengesetzten Datensatz aus den ersten beiden

Datensätzen aus Abb. 4.5.

```
@Article{Neff/94,
  AUTHOR = {Neff, C. Andrew},
  TITLE = {Specified precision polynomial root isolation is in NC},
  JOURNAL = {J. Comput.~Syst.~Sci.},
  VOLUME = {48},
  NUMBER = {3},
  PAGES = {429-463},
  YEAR = {1994},
  PUBLISHER = {Academic Press},
  ADDRESS = {New York-San Francisco-London-San Diego},
}
```

Abbildung 4.8: bibliographische Datensätze

Obiges Verfahren nimmt keine Rücksicht auf die Qualität der Inhalte. Es wird zufällig ein Datensatz als „Master"-Datensatz ausgewählt und die Daten des zu löschenden in diesen „Master"-Datensatz übernommen. Diese zufällige Auswahl des „Master"-Datensatzes kann durch eines der folgenden Verfahren ersetzt werden:

- Wähle den Datensatz aus, der zuletzt erfasst wurde. Hierbei wird davon ausgegangen, dass der neuere Datensatz eine höhere Qualität besitzt als der ältere.

- Wähle den Datensatz aus, der mehr definiert ist, das bedeutet, dass man die Anzahl der angegebenen Felder zählt und den Datensatz auswählt, der die meisten Felder enthält.

Auch das Zusammenfassen der Inhalte kann optimiert werden. Anstatt die Inhalte in den „Master"-Datensatz zu übernehmen, die in diesem nicht angegeben sind, kann man auch die Inhalte selbst auf Qualität prüfen, und den höherwertigeren Inhalt übernehmen. Bei diesem Vorgehen ist die Auswahl eines „Master"-Datensatzes unnötig, allerdings ist es sehr schwierig zu entscheiden, welcher Inhalt qualitativ höherwertiger ist. Als einfache Heuristik kann hier die Textlänge betrachtet werden: der längere Inhalt bietet eine höhere Qualität.

Wendet man eines der obigen Verfahren bei der Eliminierung der Dubletten aus der Datenmenge an, so erhält man eine neue dublettenbereinigte Menge, in der möglichst viel der Information, die in den Dubletten angegeben ist, erhalten bleibt.

Dieses Zusammenfassen der Datensätze hat den Vorteil, dass durch das Entfernen der Dubletten die Qualität des Datenbestandes nicht nur in den erfassten Daten steigt, sondern auch in der Erfassungstiefe. Unter Erfassungstiefe versteht man die Anzahl der Felder, die neben den üblichen Pflichtfeldern wie AUTHOR, TITLE, ... ebenfalls erfasst werden.

4.10 Inkrementelles Deduplizieren

Im Allgemeinen stellt ein ständig wachsender Datenbestand zwei unterschiedliche Anforderungen an die Verfahren zur Dublettenerkennung und -bereinigung. In einem ersten Schritt gilt es, in der bestehenden Datenbasis die Dubletten zu erkennen und entfernen. In der weiteren Pflege (Korrekturen und Neuerfassung) soll vermieden werden, dass sich erneut Dubletten in den Datenbestand einschleichen. Die obigen Verfahren sind in der Lage das erste Problem, die Deduplizierung der Datenbasis, zu lösen. Als Ergebnis erhält man eine dublettenbereinigte Menge. Aber wie ist es möglich effizient die Dubletten zu erkennen, wenn neue Daten in die Datenbasis eingefügt werden? Eine Möglichkeit besteht darin, die neuen Daten sofort in die Datenbasis einzufügen und anschließend die Dublettenerkennung und -bereinigung auf den gesamten Datenbestand erneut anzuwenden. Dieses Vorgehen ist allerdings nicht sinnvoll, da eine komplette Datenbereinigung ca. 2 Stunden in Anspruch nimmt und mit steigernder Größe der LEABib diese Zeit immer weiter wächst. Anzustreben wäre ein Verfahren, das es zum einen erlaubt, nur die neu erfassten Daten mit der Datenbasis auf Dubletten zu prüfen (die Datenbasis ist dublettenfrei, deshalb reicht es aus, nur die neuen Datensätze mit den bestehenden Daten zu vergleichen) oder online (bei der Erfassung eines neuen Datensatzes) die Dublettenprüfung auszuführen. In beiden Fällen ist es notwendig, bestimmte Zusatzinformationen aus der Dublettenbereinigung des Grundbestandes zu verarbeiten.

Im Folgenden wird beschrieben, wie obige Verfahren erweitert werden können, damit sie auch auf das Problem der inkrementellen Deduplizierung angewendet werden können und welche Zusatzinformationen nötig sind. Es wird vorausgesetzt, dass in den neu hinzuzufügenden Daten keine Dubletten existieren. Um das zu testen, wendet man einfach eines der obigen Verfahren auf diese Menge an.

Ein zentrales Problem stellt der Zugriff auf die einzelnen Datensätze dar. Sind diese in einer Datenbank abgelegt, so können mittels eines `SELECT * FROM data WHERE citkey IN (...)` alle Daten mit den entsprechenden CITKEYs selektiert werden. Die LEABib besteht im Gegensatz dazu aus einer Textdatei (ca. 50MB), in der die einzelnen vollständigen Datensätze sortiert nach dem CITKEY abgelegt sind. Ein Einlesen der Datei in den Hauptspeicher sollte vermieden werden, da das Einlesen der LEABib sicher mehr Zeit benötigt als die inkrementelle Deduplizierung. Aber wie können effizient aus der LEABib bestimmte Datensätze gelesen werden? Durch die Sortierung anhand der CITKEYs ist eine binäre Suche nach einem bestimmten Datensatz in Zeit $O(\log n)$ durchführbar. Eine Alternative ist, einen Index aufzubauen und zu speichern, in dem zu jedem CITKEY die Position des entsprechenden Datensatzes in der LEABib angegeben ist. Der Index ist sehr viel kleiner als die LEABib (nur ca. 900kB) und lässt sich in kurzer Zeit einlesen. Anhand dieser Informationen ist es dann möglich, effizient einen Datensatz aus der LEABib zu lesen (das Lesen eines Datensatzes dauert nur noch wenige Millisekunden).

4.10.1 Cluster-Verfahren

Das Cluster-Verfahren erstellt mit einer Funktion f die einzelnen Cluster. Innerhalb dieser Cluster wird mit einer vollständigen Suche nach Dubletten gesucht. Fügt man neue Daten hinzu, so muss für diese die Zugehörigkeit zu den Clustern bestimmt werden. Dazu wendet man die Funktion f auf die neuen Daten an und erhält somit die Zuordnung der neuen Daten zu den Clustern.
Dabei können zwei Fälle auftreten:

der Datensatz wird in einen bestehenden Cluster eingefügt Der Datensatz wird in den entsprechenden Cluster eingefügt und anschließend die Suche nach Dubletten ausgeführt.

der Datensatz definiert einen neuen Cluster Dieser Cluster enthält nur einen Datensatz und somit kann es in diesem Cluster auch keine Dubletten geben.

Welche Zusatzinformationen sind nötig, um das Einsortieren der neuen Daten in die Cluster effizient zu ermöglichen?
Zum einen werden die Schlüssel der Cluster benötigt und zum anderen die Datensätze, die in diese Cluster einsortiert wurden. Dazu legt man in einer Datei die entsprechenden Daten ab: die Clusterschlüssel, gefolgt von den CITKEYs der Datensätze, die in diesem Cluster liegen. Mit Hilfe dieses Cluster-Indizes kann dann jeder Cluster mit Hilfe der Index-Datei der LEABib effizient geladen und anschließend innerhalb der Cluster die Suche nach Dubletten ausgeführt werden.

4.10.2 Sliding Window* Verfahren

Das Sliding Window* Verfahren erzeugt für jeden Datensatz einen Schlüssel, anhand dessen alle Datensätze sortiert werden. Die sortierte Reihenfolge der CITKEYs und der entsprechende Sortierschlüssel werden wie oben als Sortierindex gespeichert. Die Deduplizierung benötigt darüber hinaus auch noch den Autorgraphen, dieser muss entweder aus den Daten der LEABib neu generiert werden, oder aber er wurde bei der letzten Deduplizierung gespeichert.
Die inkrementelle Deduplizierung erzeugt zuerst für alle neuen Daten den Sortierschlüssel und sortiert die Daten anhand diesem. Als nächstes wird der Autorengraph geladen. In diesen Autorengraphen fügt man die neuen Autoren hinzu und man erhält einen neuen Autorengraphen für die bestehenden incl. der neuen Daten (diesen Graphen[5] speichert man wieder für die nächste Deduplizierung).
Nun geht man die sortierte Liste L der neuen Daten durch, und für jeden Datensatz $d \in L$ führt man folgende Operationen aus: Setze $S = \emptyset$. Suche den nächst größeren Schlüssel im Schlüsselindex und wähle die nächsten $k - 1$ (k = Fenstergröße) Schlüssel aus dem Schlüsselindex aus und füge diese zu S hinzu. Füge die Daten aus dem Autorengraphen

[5] Der Graph enthält ca. 250.000 Knoten und ca. 500.000 Kanten, also bietet sich in diesem Falle eine Speicherung in der Form Knotenliste + Kantenliste an.

zu S hinzu. Lade alle Daten aus S. S enthält jetzt alle Daten aus der LEABib, die mit d verglichen werden müssen.

4.11 Zusammenfassung

Die Suche nach Dubletten erfordert neben einer geeigneten Vergleichs-Methode effiziente Verfahren, die die Anzahl der Vergleiche möglichst gering halten. Ein Cluster-Verfahren bietet sich an, wenn sichergestellt werden kann, dass die Methode zum Erzeugen der Cluster robust gegenüber den auftretenden Fehlern ist. Im Fall der LEABib ist so eine Funktion zum Erzeugen der Cluster gegeben. In vielen Anwendungen ist allerdings diese Robustheit nicht gegeben, so dass hierfür das Sliding Window* Verfahren besser geeignet ist.

Kapitel 5

Ähnliche Zeitschriften

Oftmals sind einem Leser nicht immer alle relevanten Zeitschriften oder Konferenzen bekannt, in denen Artikel zu seinem Interessensgebiet veröffentlicht werden. Ein Tool, das anhand weniger ausgewählter Veröffentlichungen, Journale oder Konferenzen weitere zum Thema relevante Journale und Konferenzen bestimmt, kann hierbei den Leser unterstützen.

Die manuelle Erfassung von bibliographischen Daten erfordert einen hohen Zeitaufwand. Aus diesem Grund werden meist nicht alle verfügbaren Daten erfasst, wie die Angabe der Keywords und Abstracts. Um trotzdem den einzelnen Datensätzen Themen zuordnen zu können, ist es notwendig, aus den gegebenen Daten entsprechende Schlüsselwörter automatisch abzuleiten. Auf Basis dieser Schlüsselwörter und den gegebenen kann die Suche nach ähnlichen Journalen erfolgen.
Dieses Kapitel gibt zuerst einen kurzen Überblick über bestehende Verfahren, um aus einer Sammlung von Dokumenten ähnliche Dokumente zu bestimmen. Im Anschluss daran wird ein weiteres Verfahren vorgestellt, das bei der Bestimmung relevanter Dokumente nur eine Teilmenge der Sammlung betrachtet und somit schneller ein Ergebnis liefert. Zum Schluss wird darauf eingegangen, wie die bestehenden Daten mit Schlüsselwörtern angereichert werden und wie diese behandelt werden können.
Zum Abschluss wird ein interaktives Verfahren zur Klassifikation der LEABib-Daten vorgestellt.

5.1 Information Retrieval

Im Bereich des Information Retrieval wurden unterschiedliche Verfahren entwickelt, um aus einer Menge von Dokumenten die zu einer Anfrage a relevanten Dokumente zu finden.
Im Allgemeinen kann man die verschiedenen Verfahren in zwei Klassen einteilen:

- Boolesche Methode
- Vektor-Modell-Methode

Zuerst einige Definitionen, die im Folgenden häufiger verwendet werden:

d_i bezeichnet das i-te Dokument

M bezeichnet die Menge der Dokumente: $M = \bigcup_i d_i$

t_{k,d_i} bezeichnet den k-ten Term im Dokument d_i

T_{d_i} bezeichnet die Menge der Terme im Dokument d_i: $T_{d_i} = \bigcup_k t_{k,d_i}$

T_M bezeichnet die Gesamtmenge der Terme: $T_M = \bigcup_{d \in M} T_{d_i}$

5.1.1 Boolesche Methode

Eine boolesche Anfrage besteht aus den Operatoren $\vee, \wedge, \neg$. Mit diesen Operatoren werden Anfragen gebildet, die die Existenz von Termen in den Dokumenten fordern oder ausschließen.

Beispiel:
Die Anfrage „information $\wedge$ retrieval $\wedge\neg$ database“ findet alle Dokumente, in denen die Terme „information“ und „retrieval“ vorkommen, und der Term „database“ nicht enthalten ist.

□

Bei der *booleschen Methode* kann sofort entschieden werden, ob ein Dokument zur Anfrage relevant ist, indem die Anfrage auf ein Dokument in der Menge angewendet wird. Diese Methode eignet sich auch, um eine bestimmte Teilmenge an Dokumenten zu selektieren. Zunächst definiert man einen Ausdruck s_i für jedes Dokument, der nur dieses selektiert. Der Term

$$s_1 \vee s_2 \ldots \vee s_n$$

selektiert dann genau die gewünschten Dokumente aus der Menge.
Der Nachteil der booleschen Methode ist, dass die Relevanz eines Dokumentes nur 0 (=nicht relevant) oder 1 (=relevant) beträgt. Deshalb kann dieser Wert nicht verwendet werden, um die gefundenen Dokumente zu ranken. Ebenso ist die Anzahl der relevanten Dokumente nicht vorhersagbar. Weist man den einzelnen Termen der Anfrage Gewichte zu, so können diese Gewichte bei der Bestimmung der Relevanz eines Dokumentes zur Anfrage berücksichtigt werden. Dieses Vorgehen führt dann zur Fuzzy-Methode.

5.1.2 Vektor-Modell

Das Vektor-Modell ordnet jedem Dokument einen Vektor $\vec{v_{d_i}}$ zu, wobei $\vec{v_{d,k}}$ (die Komponente für den k-ten Term) die Relevanz des Terms zum Dokument d angibt. 0 bedeutet entweder, der Term ist im Dokument d nicht enthalten oder nicht relevant. Der Vektor $\vec{v}$ besteht aus n Komponenten, wobei n die Gesamtanzahl der Terme aller Dokumente ist. Bei einer großen Anzahl von Dokumenten kann die Menge der Terme und somit auch n sehr groß werden. Die Terme und damit auch die Dokument-Vektoren $\vec{v_d}$ und der entsprechende Vektorraum V werden in einer Initialisierungsphase (dem Indizieren der Dokumente) bestimmt.

Stellt ein Benutzer eine Anfrage an das System, so wird zunächst die Anfrage auf den Vektor $\vec{a} \in V$ abgebildet. Anschließend erfolgt die Suche nach Dokumenten, deren Dokument-Vektor $\vec{v_d}$ ähnlich zum Anfrage-Vektor $\vec{a}$ ist, also relevant zur Anfrage sind. In einem anschließenden Schritt erfolgt das Ranking der ermittelten Dokumente.

Mögliche Methoden zum Bestimmen der relevanten Dokumente sind z.B.: alle Dokumente innerhalb einer ϵ-Umgebung des Anfrage-Vektors, oder der $cos(\angle(v_{d_1}, v_{d_2}))$, der Winkel zwischen Anfrage- und Dokument-Vektor, d.h alle Dokumente, deren Vektor $\vec{v_d}$ einen kleinen Winkel zum Anfrage-Vektor $\vec{a}$ besitzen.

Als Ranking-Methode kann jede Metrik im gegebenen Vektorraum V dienen. Ist die Methode zum Bestimmen der relevanten Dokumente bereits eine Metrik, so können die Ergebnisse bei der Auswahl der relevanten Dokumente bei der Bestimmung des Rankings verwendet werden.

Im Gegensatz zur booleschen Methode kann im Vektor-Modell die Menge der relevanten Dokumente erst am Ende ausgegeben werden, da das Ranking der Dokumente vorher noch nicht bestimmt ist.

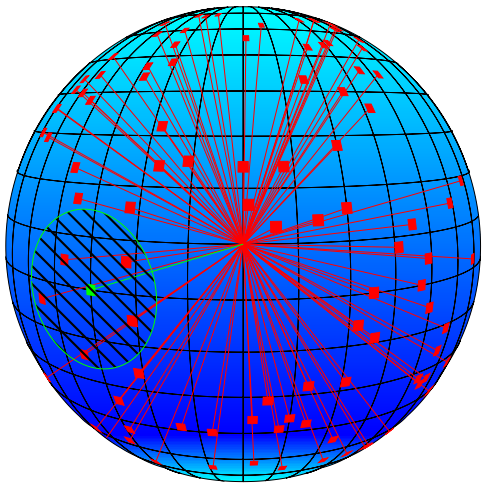


Abbildung 5.1: Anfrage- und Dokument-Vektoren

Beispiel:
Abb. 5.1 zeigt die Dokumente im Vektorraum. Jeder rote Punkt stellt ein Dokument dar, die Vektoren die einzelnen Dokument-Vektoren. Der grüne Vektor stellt die Anfrage dar. Alle Dokumente innerhalb des grünen Kreises sind die passenden Dokumente zur Anfrage. Wird der Vektor $\vec{d_i}$ eines Dokuments d_i mit dem Anfrage-Vektor $\vec{a}$ verglichen, so wird der $cos(\angle(\vec{d_i}, \vec{a}))$, der Winkel zwischen den beiden Vektoren $\vec{d_i}$ und $\vec{a}$ berechnet. Zwei Dokumente sind umso „ähnlicher“, je geringer der Winkel zwischen den beiden Vektoren ist. Der $cos(\angle(\vec{d_i}, \vec{a}))$ liegt zwischen 0 und 1, somit lässt sich der Wert auch für das Ranking der Ergebnisse verwenden, je näher der Wert an 1 liegt, um so relevanter ist das Dokument.
Diese Dokumente werden anhand ihres Abstandes zur Anfrage ($cos(\angle(v_{d_1}, a))$ absteigend sortiert ausgegeben.

□

5.2 Publikationsgraph

Der Publikationsgraph besteht aus Knoten für

- Publikationen P
- Autoren A
- Journale, Serien, Konferenzen V
- Keywords K

Die Kanten sind wie folgt definiert:

Autor a_i - Publikation p_j: Der Autor a_i ist Autor der Publikation p_j

Publikation p_i- Journal, Serie, Konferenz v_j: Die Publikation p_i wurde im Journal, der Serie oder Konferenz v_j veröffentlicht

Publikation p_i - Keyword k_j: Die Publikation p_i enthält das Keyword k_j

Mittels obiger Definition des Publikationsgraph erhält man einen „bipartiten“ Graphen, falls man immer nur eine Relation betrachtet, wobei man die Knotenmenge in vier verschiedene Mengen unterteilen kann.

$V_1 = P$ Autoren

$V_2 = A$ Publikationen

$V_3 = V$ Journale, Serien, Konferenzen

$V_4 = K$ Keywords

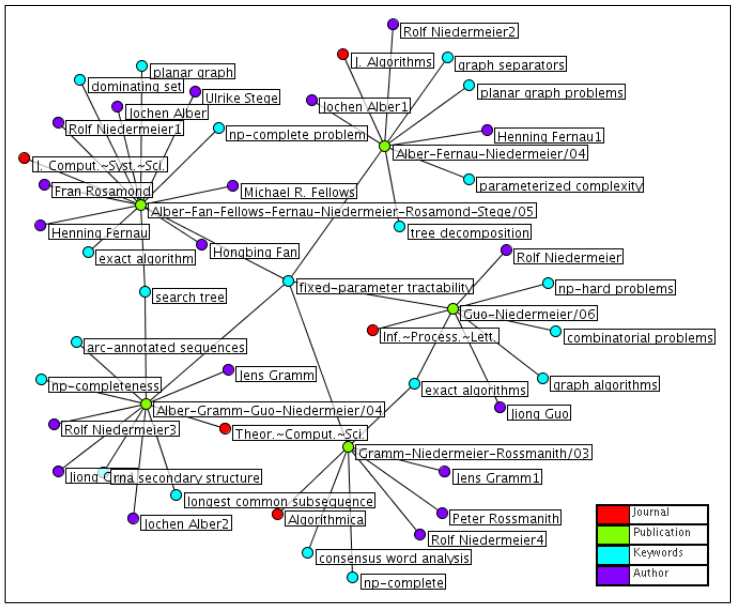


Abbildung 5.2: Publikationsgraph

5.3 Struktur

In Abbildung 5.2 kann man erkennen, dass die beiden Publikationen p_1 und p_2 aus den beiden Journalen v_1 und v_2 zwei gemeinsame Keywords besitzen. Im Gegensatz dazu besitzen die beiden Publikationen p_1 und p_4 aus den beiden Journalen v_1 und v_3 keine gemeinsamen Keywords.

Zwei Journale behandeln das gleiche Thema, wenn die Artikel in den Journalen viele gemeinsame Keywords besitzen. Je mehr gleiche Keywords die Artikel besitzen, desto „ähnlicher“ sind die Themen der Journale.

5.4 Gewichtung der Keywords

5.4.1 Definitionen

Definition 5.1: Termhäufigkeit f
Die Termhäufigkeit $f_{t,d}$ gibt an, wie oft der Term f_t im Dokument d vorkommt. $f_{t,d}$ kann mittels $\max\{f_{t,d}\}$ normalisiert werden $\rightarrow f'_{t,d} = \frac{f_{t,d}}{\max\{f_{t,d}\}}$

Definition 5.2: (invertierte) Dokumenthäufigkeit $(i)df$
Die Dokumenthäufigkeit df_t gibt an, in wie vielen Dokumenten der Term t vorkommt. Die

invertierte Dokumenthäufigkeit berechnet sich wie folgt:

$$idf_t = \log \frac{N}{df_t}$$

wobei N die Gesamtanzahl der Dokumente bezeichnet.

Definition 5.3: Termfrequenz tf
Die Termfrequenz tf gibt die relative Häufigkeit eines Wortes beziehungsweise Termes in einem gesamten Dokument an.

$$tf_{t,d} = \frac{f_{t,d}}{n(d)}$$

wobei $n(d)$ die Anzahl der Terme im Dokument d bezeichnet.

Definition 5.4: Gewichtung einzelner Keywords

$$c_{t,d} = f'_{t,d} idf_t = \frac{f_{t,d}}{\max\{f_{t,d}\}} \log \frac{N}{df_t}$$

Durch eine geeignete Gewichtung kann die Menge J entsprechend sortiert werden und es sind somit die „besseren“ Journale an erster Stelle. Als geeignete Gewichtung bietet sich die Summe gleicher Keywords an, wobei die Keywords allerdings mit der Termfrequenz multipliziert werden.

5.5 Algorithmus

Die Bestimmung ähnlicher Journale, ausgehend von einzelnen vorgegebenen Artikeln oder Journalen, erfolgt in zwei Schritten:

- Breitensuche von den selektieren Artikeln oder Journalen nach Keywords (beschränkt auf max. Tiefe 2) → Menge K von Keywords.
- Breitensuche von jedem Keyword k aus K nach Journalen (beschränkt auf max. Tiefe 2) → Menge J von Journalen.

Die Menge J enthält die zum gegebenen Journal J_i „ähnlichen“ Journale.

5.6 Probleme

Der Publikationsgraph besteht aus einer großen Anzahl von Zusammenhangskomponenten. Diese „Zerteilung“ des Graphen macht es sehr schwierig, plausible Aussagen über ähnliche Journale zu treffen, da zum einen viele Journale nicht über gemeinsame Keywords verbunden sind und zum anderen die Gewichtung der Keywords nicht vollständig berechnet werden kann, da für eine große Anzahl von Publikationen diese nicht angegeben sind.

5.6.1 Keywords aus dem Titel extrahieren

Bei der Erfassung von bibliographischen Daten ist es sehr zeitaufwendig, die Keywords für jeden zu erfassenden Datensatz einzugeben. Oftmals sind keine Keywords vom Verfasser angegeben, so dass es nicht möglich ist, diese zu erfassen. Nur mit Hilfe des Volltextes und Fachkenntnis wäre es möglich bei diesen Publikationen entsprechende Keywords zu vergeben. Da dies allerdings einen sehr hohen Zeitaufwand bei der Erfassung bedeuten würde, kann dies nicht bewerkstelligt werden. Somit bleibt die Frage offen: Wie können aus den gegebenen Daten geeignete Keywords gewonnen werden?
Die einzige Anlaufstelle zur automatischen Gewinnung von Keywords ist der Titel der Publikation. Jedes einzelne Wort im Titel könnte man als Keyword interpretieren. Allerdings besteht ein Titel neben aussagekräftigen auch aus nicht aussagekräftigen Teilen. Diese sind mit geeigneten Methoden auszufiltern. Ein weiteres Problem besteht darin, dass Keywords, die von Autoren vergeben wurden und somit evtl. mit erfasst wurden, aussagekräftiger sind, als automatisch generierte.

Beispiel:

```
@article{Watkins/05,
      ...
      title = {Product eigenvalue problems},
      keywords = {eigenvalue, GR algorithm, generalized eigenvalue problem, SVD,
            symmetric„ skew symmetric, positive definite, totally positive, hamiltonian,
            pseudosymmetric, symplectic, unitary}
}

@article{Burq-Zworski/05,
      ...
      title = {Bouncing ball modes and quantum chaos},
      keywords = {eigenfunctions, billiards, quantum ergodicity}
}
```

□

Im ersten Artikel würde ein naiver Algorithmus die drei Keywords „product“, „eigenvalue“ und „problems“ vergeben. Betrachtet man aber die Keywords, die der Autor selbst angegeben hat, so ist nur eines der drei Keywords enthalten. Die beiden anderen finden sich nicht in der Liste. Das gleiche Problem tritt auch im zweiten Artikel auf. Ein weiteres Problem stellen Wörter wie „and“, „the“, „for“, ... dar. Diese tragen nichts zum Inhalt des Titels bei. Aber wie können diese Probleme umgangen werden? Vier verschiedene Methoden erlauben es, die Qualität der automatisch erzeugten Keywords aus den Titeln zu erhöhen:

- Stoppwortlisten
- Cut-Off seltener Keywords

- Stemming
- Zuverlässigkeit der Keywords

Stoppwortlisten In einer Stoppwortliste sind alle Wörter gespeichert, die ignoriert werden sollen. Der Algorithmus prüft für jedes einzelne Wort aus dem Titel, ob es in der Stoppwortliste enthalten ist, falls ja, wird dieses Wort nicht in die Keywords aufgenommen. Da es sich meist um englischsprachige Publikationen handelt, ist die Verwendung einer englischen Stoppwortliste sinnvoll [STN]. Auf diese Weise reduziert sich die Anzahl der Keywords um ca 25%.

Cut-Off seltener Keywords Ein weiteres Problem stellen Keywords dar, die nur sehr selten verwendet werden. Diese werden vom Ranking-Algorithmus als sehr wichtig betrachtet und somit alle gefundenen Dokumente entsprechend hoch bewertet. Nicht häufig vorkommende Keywords, die von den Autoren vergeben werden, sind aussagekräftig, automatisch generierte dagegen weniger. Diese würden das Ergebnis verfälschen. Aus diesem Grund entfernt man ebenfalls alle selten vorkommenden Keywords, die durch die Verarbeitung der Titel gewonnen wurden wieder aus den Keywords. Bei einem Cut-Off von 10% erhält man einen relativ zusammenhängenden Graphen.

Stemming Betrachtet man den ersten Artikel im Beispiel, so wird „problems“ zu den Keywords hinzugefügt. Da es sich hier nicht um die Grundform handelt, wird sich selten eine Überschneidung mit den bestehenden Keywords ergeben. Somit ist dieses generierte Keyword kein geeigneter Term, um als Keyword für den Artikel aufgenommen zu werden. Abhilfe schafft hier Stemming. Beim Stemming reduziert man die Wörter auf ihre Grundform. In obigem Beispiel „problems“ auf „problem“. Ein bekannter Stemming-Algorithmus ist der Porter-Stemmer [Por97].

Zuverlässigkeit der Keywords In den obigen Abschnitten wurden die Probleme und geeignete Methoden zur Lösung gezeigt, die bei der automatischen Keywordvergabe aus den Titeln der Publikationen entstehen. Trotz dieser Verfahren sind automatisch generierte Keywords nicht so aussagekräftig und zuverlässig wie solche, die von den Autoren selbst vergeben werden. Aus diesem Grund ist es nötig, automatisch generierte Keywords geringer zu bewerten als vorgegebene. Um diesem Umstand Rechnung zu tragen, werden die Bewertungen der automatisch generierten Keywords mit einer Konstanten α ($0 \leq \alpha \leq 1$) multipliziert. Für $\alpha = 0$ werden diese Keywords ignoriert, für $\alpha = 1$ als gleichwertig betrachtet.

Beispiel:
J. of Graph Theory (51)
- Graphs and Combinatorics
- SIAM J. Comput.
- J. Comb. Theory Series A
- SIAM J. Algebraic Discrete Methods
- J. Symbolic Computation
- SIAM J. Disc. Math.
- Acta Inf.
- Int. Journal of Algebra and Computation
- IEEE Trans. Comput.
- J. Comb. Theory

LNCS (252)
- Interdisciplinary Information Sciences
- Comput. Methods Appl. Mech. Eng.
- Computer Architecture News
- Invent. Math.
- Adv. Math., Beijing
- SIAM J. Appl. Math.
- DIMACS — Series in Discrete Mathematics and Theoretical Computer Science
- Bull. Soc. Math. France
- J. of Graph Theory
- Colloq. Math.

Zum Journal **J. of Graph Theory** wurden 51 ähnliche Zeitschriften gefunden. In der Liste findet man Journale, die ähnliche Probleme behandeln, wie z.B. *Graphs and Combinatorics*. Betrachtet man das Journal **LNCS**, so befinden sich in den 252 ähnlichen Journalen die unterschiedlichsten Themenbereiche. Das liegt daran, dass das Journal LNCS nicht ein spezielles Thema der Informatik behandelt, sondern ein breites Spektrum abdeckt und deshalb in der Liste die verschiedensten Journale gelistet sind.

□

5.7 Interaktive Klassifikation

Für die Informatik bietet ACM[1] im Internet[2] ein Klassifikations-Schema an, mit dessen Hilfe sich die Veröffentlichungen kategorisieren lassen. Die Einordnung einer Liste von Pu-

[1] Association for Computing Machinery
[2] http://www.acm.org/class/

blikationen in dieses Klassifikations-Schema ist für Laien aufgrund fehlender Fachkenntnis und auch für „Insider“ wegen des hohen Aufwands kaum manuell durchführbar. In den meisten Fällen weisen nicht einmal die Autoren ihrer Veröffentlichung die entsprechenden Klassifikationen zu. In großen Datenbeständen wie der LEABib wäre es allerdings wünschenswert, neben den vorhandenen Möglichkeiten durch die Daten zu navigieren, auch auf Basis einer Klassifikation die Daten zu selektieren. Da aber in der LEABib keine Klassifikationen erfasst wurden bzw. werden, ist ein automatischer Klassifizierer nötig, der die bestehenden Daten um diese anreichert. In der Literatur gibt es unterschiedliche Modelle, die es erlauben, Daten zu klassifizieren. Bei der Klassifizierung von Daten ergeben sich zwei unterschiedliche Fragestellungen:

- ist der Datensatz d in die Klasse c_i einzuordnen?
- in welche der Klassen $c_1, c_2, \ldots, c_k$ ist der Datensatz d einzuordnen?

Im weiteren wird davon ausgegangen, dass das Klassifikations-Schema bekannt ist und entsprechende Trainingsdaten vorliegen. Im Falle der LEABib dient als Klassifikations-Schema die ACM-Klassifikation und die Trainingsdaten sind in der LEABib bereits klassifiziert.
Es gibt verschiedene Verfahren, die eine automatische Klassifikation anhand von Trainingsdaten erlauben. Dazu zählen unter anderem der Bayes-Klassifikator und das Vektor-Modell. Das Bayes-Verfahren baut auf dem Theorem von Bayes auf, das wie folgt definiert ist: Für zwei Ereignisse A und B gilt:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Dabei ordnet das Verfahren jedem Datensatz die Klassifikation zu, die „am Besten passt“.
Die Funktionsweise des Vektormodells wurde bereits in Abschnitt 5.1.2 erläutert. Anhand der Trainingsdaten und mit Hilfe der Java-Bibliothek *Classifier4J*[3] wurde eine Klassifizierung der LEABib mit der Klasse *VectorClassfier* durchgeführt. Die Methode *classify(String class, String s)* von *VectorClassfier* bestimmt einen Wert zwischen 0 und 1, wobei ein Wert nahe 1 angibt, dass dieser Datensatz in die Klasse *class* passt. Als Trainingsdaten dienen die Keywords der bereits klassifizierten Datensätze, da aber nur ein Teil der Datensätze Keywords enthält, wurden in einem ersten Schritt die restlichen Daten mit dem Verfahren in Abschnitt 5.6.1 mit entsprechenden Keywords angereichert.
Der Klassifizierer bestimmt für jeden Datensatz einen Score, der angibt, wie gut dieser zur angegebenen Klasse passt. Der Klassifizierer wurde zuerst mit den Trainingsdaten initialisiert. Anschließend wird für jeden Datensatz geprüft, in welche Klassen dieser eingeordnet werden kann. Diese möglichen Klassen werden als Kandidaten in einer Tabelle mit ihren entsprechenden Score-Werten abgespeichert, wobei allerdings nur diese Kandidaten gespeichert werden, deren Score größer als α ist. Im nächsten Schritt erfolgt die Zuordnung zu den Klassen. Hierbei wird der Datensatz nicht allen gefundenen Klassen zugeordnet, son-

[3] http://classifier4j.sourceforge.net/

dern nur den maximal besten drei und sich der Score der besten und der schlechtesten noch ausgewählten Klasse nur um einen Faktor β unterscheidet.

Algorithmus 5.1: Classifier

1. Search for trainingdata $\rightarrow$ (*classes*, *trainingdata*)
2. Generate VectorClassifier $\rightarrow$ *classifier*
3. Send *trainingdata* for *classes* to *classifier*
4. **forall** *d* **in** *data* ← Iterate over all data
   - *categories* = ()
   - **forall** *c* **in** *classes* ← Iterate over all classes
     - *c* = get score for *d* and class *c*
     - **if** *score* $\geq \alpha$ **then**
       - add class *c* to *categories* with score *c*
     - **fi**
   - **od**
   - Sort *categories* by *c* in descending order
   - *max* = score of *categories*[0] ← Get max score
   - **for** *i* = 1 to 3 **do** ← Get best three classes
     - **if** score of *categories*[*i*] $\geq max * \beta$ **then**
       - add class of *categories*[*i*] to *d*
     - **fi**
   - **od**

od

5.8 Zusammenfassung

Mit Hilfe obiger Algorithmen ist es möglich, zu den Zeitschriften in der LEABib Journale oder Konferenzen zu bestimmen, die das gleiche oder ein ähnliches Themengebiet behandeln. Somit kann ein Benutzer auf weiterführende Veröffentlichungen hingewiesen und bei der Suche nach relevanten Artikeln zu einem Themengebiet unterstützt werden.
Da in bibliographischen Daten oftmals die Keywords fehlen ist eine automatische Anreicherung der Daten mit Keywords nötig. Durch den Einsatz von Stoppwortlisten, Stemming und des Cut-Offs kann die Anzahl gering und die Qualität der automatisch generierten Keywords hoch gehalten werden. Der Nutzen allerdings ist sehr hoch, da in den ursprünglichen Daten (Graphen) mehr als 16.000 Zusammenhangskomponenten existierten, in den bearbeiteten Daten nur noch ca 560. Die Anzahl der Zusammenhangskomponenten lässt sich auch durch „Verschlechterung“ der Qualität der Keywords (weniger Stoppwörter, höherer Cut-Off) nur minimal erhöhen. Bei einem Cut-Off von 10% erhält man eine geringe Anzahl von automatisch generierten Keywords und einen relativ zusammenhängenden Graphen.

Die maximale Komponente im Graphen umfasst 111.117 Knoten, wobei der gesamte Graph 112.082 Knoten enthält. Die restlichen Komponenten sind sehr klein (≤ 20), wobei deren Anzahl sehr gering ist (siehe Tabelle C.2). In der originalen LEABib existieren neben dem größten Cluster (72800 Knoten) weitere große Cluster.
Neben der Suche nach ähnlichen Zeitschriften können die so gewonnenen Keywords ebenso für die automatische Klassifikation verwendet werden. Nach einer initialen Trainingsphase können den Daten mit einem Klassifizierer die entsprechenden Klassifikationen zugewiesen werden.

Kapitel 6

Autorenerkennung

In den vorhergehenden Abschnitten wurden Verfahren gezeigt, die doppelte Datensätze erkennen und eliminieren und zu einem vorgegebenen Artikel oder Journal ähnliche Dokumente finden. Diese Verfahren nutzen vor allem syntaktische Methoden um festzustellen, ob zwei Objekte ähnlich sind. Diese syntaktischen Methoden funktionieren bei der Erkennung der Autoren nur noch bedingt, da bei der Erfassung der Autoren unterschiedliche Fehlerquellen existieren.
Die Fehlerquellen lassen sich in drei Kategorien einteilen:

Erfassungsfehler sind Tippfehler oder auch Fehler in der Angabe der Akzentzeichen.

Fehlerhafte Angaben sind die häufigste Fehlerquelle. In vielen Fällen wird derselbe Autor unterschiedlich angegeben. Oftmals werden die Vornamen abgekürzt oder bei mehreren Vornamen nur der erste genannt. Ein weiteres Problem stellen Akzentzeichen dar. Diese werden einfach weggelassen oder durch falsche ersetzt (Donald Ervin Knuth, Donald E. Knuth, Lopez, López)

Umbenennungen sind die größte Herausforderung an eine Autorenerkennung. Bestimmte Autoren ändern ihren Vor- oder Nachnamen. Vor allem nicht-amerikanische Autoren publizieren zuerst unter ihrem richtigen Namen und „amerikanisieren“ später diesen (z.B. Paweł zu Paul, Janusz zu John).

Die Erkennung der Autoren ist auf Grund der obigen Fehlerquellen auf weitere Daten und Verfahren angewiesen, um mit hoher Zuverlässigkeit die einzelnen Autoren richtig zu identifizieren. Das Problem bei der Identifizierung der Autoren stellen allerdings die Daten dar. Um eine hohe Treffsicherheit zu gewährleisten sind Zusatzinformationen nötig, die zum einem aus den bestehenden Daten gewonnen werden können, zum anderen durch dritte auf Nachfrage bereitgestellt werden müssen.
In [HEG06] stellen Jian Huang, Seyda Ertekin und C. Lee Giles ein Verfahren vor, das mit Hilfe von DBSCAN [EKSX96] und Support Vector Machines [SS01] die einzelnen Autoren unterscheiden kann.

In diesem Kapitel wird ein clusterbasiertes, iteratives Verfahren gezeigt, das mit hoher Zuverlässigkeit die einzelnen Autoren in einer bibliographischen Datenbank identifiziert. Dabei verwendet das Verfahren nur solche Daten, die bereits in der Datenbasis verfügbar sind und leitet aus diesen neue Daten ab, die eine hohe Treffsicherheit bei der Identifizierung gewährleisten.

Im ersten Teil werden die Beziehungen betrachtet, die durch die Daten definiert sind, wie z.B. die Mitautorenbeziehung. Aus diesen Beziehungen lassen sich unterschiedliche Eigenschaften ableiten: die Autoren a und b sind Kandidaten für Dubletten oder können nicht identisch sein. Darüber hinaus ist es möglich, den Autoren Themengebiete zuzuordnen, sowie Zeiträume zu bestimmen, in denen bestimmte Autoren bzw. Autorengruppen publizieren. Alle diese Daten werden zur Bestimmung der Ähnlichkeit zweier Autoren a und b herangezogen.

6.1 Autorengraph

6.1.1 Publikationsgraph

Die bibliographischen Daten definieren verschiedene Beziehungen zwischen den einzelnen Datensätzen. Das Feld AUTHOR definiert die Beziehung *Publikation p wurde von den Autoren $a_1, a_2, \ldots a_n$ verfasst.* Durch die Angabe der Keywords ist es möglich, den Autoren und Journalen einzelne Themen zuzuordnen. Auf Basis der Daten ist es möglich, verschiedene Beziehungen unter den Daten zu bestimmen. In Abb. 6.1 sind drei solcher Beziehungen in einem Graphen dargestellt (a ist Autor der Publikation p, Publikation p wurde veröffentlicht in j, Publikation p enthält die Keywords $k_1, \ldots, k_j$).

6.1.2 Mitautoren-Beziehung

Eine zentrale Rolle bei der Identifizierung der Autoren spielt die Mitautoren-Beziehung. Diese ist definiert durch das Feld AUTHOR, wobei alle Autoren einer Publikation als eine Autorengruppe definiert werden. In Abb. 6.1 sind die Mitautoren eines Autors a durch die gemeinsame Publikation p gegeben. Abb. 6.2 zeigt die Mitautoren-Beziehung in einem Graphen, wobei die einzelnen Autorengruppen durch vollständig verbundene Teilgraphen dargestellt sind.

Die Aufgabe der Autorenidentifizierung ist es, identische Autoren in diesem Graphen zu verbinden. Identische Autoren sind Autoren, die die gleiche Person darstellen. Stellen zwei gleiche Schreibweisen unterschiedliche Autoren dar, so soll dies ebenso erkannt werden, wie der Fall, dass zwei unterschiedliche Schreibweisen den gleichen Autor bezeichnen. Um diese Identifizierung der Autoren durchzuführen und eine hohe Zuverlässigkeit zu erreichen, sind neben dem Mitautorengraphen weitere Daten in der Verarbeitung nötig.

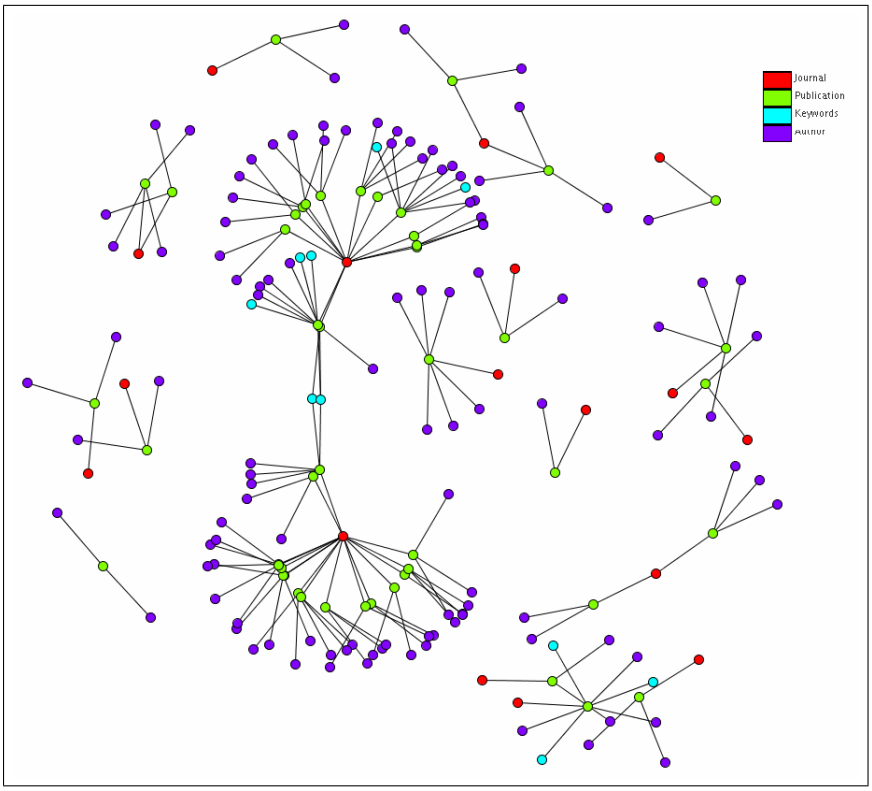


Abbildung 6.1: Publikationsgraph

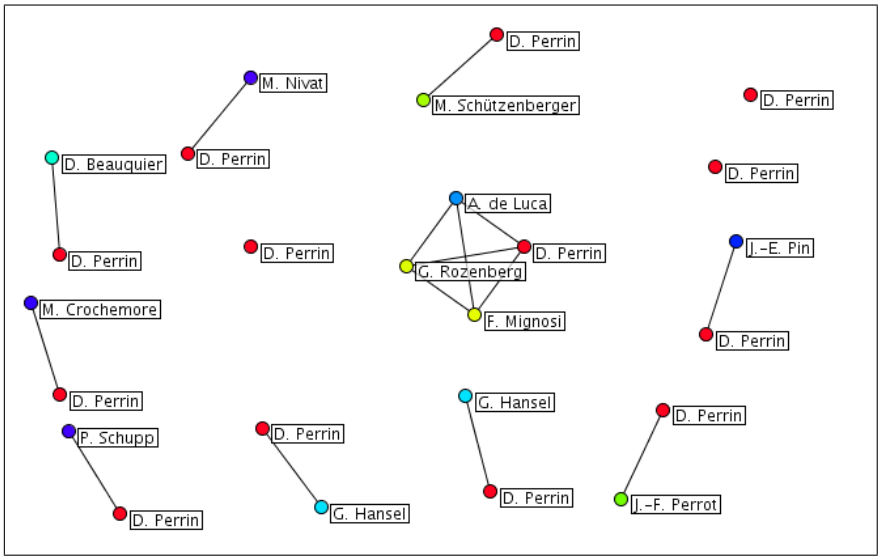


Abbildung 6.2: Graph der Mitautoren

Mögliche und auszuschließende Dubletten

Bei der Verarbeitung des Mitautorengraphen können die Autoren in zwei unterschiedliche Kategorien eingeteilt werden. Eine Kategorie definiert die möglichen Dubletten, die andere bezeichnet die Autoren, die nicht in der Dublettensuche berücksichtigt werden müssen, da diese keinesfalls den gleichen Autor bezeichnen.
Besteht eine Autorengruppe aus den Autoren $a_1, \ldots, a_k$, so können die Autoren $a_i | i \neq j$ nicht den Autor a_j bezeichnen, da bei der Angabe der Autoren einer Publikation jeder Autor nur einmal angegeben wird und somit alle angegebenen Autoren einer Publikation unterschiedliche Autoren bezeichnen.
Die Menge der potenziellen Dubletten a_i von Autor a einer Publikation p ist gegeben durch die Menge aller Autoren abzüglich der Menge der Mitautoren von a in p.
Die LEABib enthält mehr als 82.300 bibliographische Einträge und mehr als 168.000 Autoren. Im Durchschnitt verfassen ca. zwei Autoren eine Publikation, wobei die Anzahl der Publikationen, die von zwei Autoren verfasst wurden ca. 30.300 beträgt[1]. Weitere 30.300 Publikationen wurden nur von einem Autor verfasst. Die maximale Anzahl Autoren einer Publikation ist 62. Diese Zahlen belegen, dass eine nur auf der Mitautoren-Beziehung basierende Autorenidentifizierung viele Dubletten nicht erkennt, da ca. 30.300 Autoren (18%) eine Publikation alleine verfassten und somit auch keine Mitautoren haben. Diese Einzelautoren sind im Mitautorengraphen isolierte Knoten, die keine Verbindung zu anderen Autoren besitzen und somit auch bei der Identifizierung über die Mitautorenbeziehung nicht behandelt werden können. Aus diesem Grund ist es nötig, weitere Daten abzuleiten und bei der Identifizierung zu verwenden.

6.2 Autorenidentifizierung

In Abschnitt 6.1.2 wurde gezeigt, dass die Identifizierung der Autoren rein auf dem Mitautorengraphen nicht zuverlässig arbeiten kann und dass weitere Daten aus den bestehenden abgeleitet werden müssen, um die Autoren zuverlässig zu erkennen. Aber welche Daten sind dazu geeignet und wie können diese abgeleitet werden? Im Folgenden betrachten wir die unterschiedlichen Felder, die die Identifizierung der Autoren weiter unterstützen. Dazu zählen die Keywords, das Publikationsjahr und die Institution.

Das Feld AUTHOR

Die wichtigste Eigenschaft bei der Identifizierung der Autoren ist der Name des Autors. In der LEABib erfolgt die Angabe der Autoren durch folgende Regeln [Kir]:

- <Nachname>
- <Nachname>[, Jr.]

[1] siehe Anhang A, Seite 89

- <Nachname>[, <Vorname>]
- <Nachname>[, <Vorname>, Jr.]

Vom eigentlichen Nachnamen durch ein Leerzeichen abgetrennt darf angegeben werden, der wie vielte seines Namens der Autor ist, z.B. ist *Müller III, Lieschen* ein zulässiger Name. Desweiteren sind folgende Regeln zu beachten:

- Alle Namen sind durch „ and “ voneinander zu trennen.
- Im Vornamen sind sowohl mehrere ausgeschriebene als auch mehrere abgekürzte Namen zugelassen. Abgekürzte Namen haben direkt aufeinander zu folgen. Korrekt ist z.B. `M.U. Carsten Kurt` als Vorname, wohingegen `M. U. Carsten Kurt` falsch ist.
- LaTeX Befehle sind an jeder Stelle in Namen zugelassen.

Ein Problem beim Vergleich zweier Autorennamen besteht darin, dass bei ein und dem selbem Autor oftmals der Name unterschiedlich angegeben ist. Die folgenden Probleme treten dabei auf:

fehlende Vornamen In manchen Fällen wird bei mehreren Vornamen nur der erste genannt, die restlichen werden nicht mit angegeben

abgekürzte Vornamen Oftmals werden alle Vornamen oder nur ein Teil als Initialen angegeben

LaTeX-Befehle und Akzentzeichen Da in den Namen LaTeX-Befehle erlaubt sind, erfolgt die Angabe oftmals in unterschiedlicher Form. Ebenso werden Akzentzeichen oft falsch oder gar nicht angegeben.

Um diese Probleme zu umgehen, erfolgt vor dem Vergleich eine Normalisierung der Namen.

- Entferne LaTeX-Befehle
- Ersetze Buchstaben mit Akzentzeichen durch den Buchstaben ohne Akzentzeichen ($\acute{e} \rightarrow e$)
- Ersetze alle Vornamen durch die Initialen (*M.U. Carsten Kurt* $\rightarrow$ *M.U.C Kurt*)
- Entferne die Namenszusätze, wie z.B. *Jr.* oder die Zählung
- Erzeuge die Normalform in der Form <Nachname>, <Initialen>

Auf Basis der normalisierten Namen kann ein Vergleich zwischen zwei Namen erfolgen: Stimmen die Nachnamen nicht überein, so handelt es sich um verschiedene Autoren, stimmen dagegen neben den Nachnamen auch die Initialen überein, so handelt es sich eventuell um den selben Autor.

Beispiel:

M.U. Carsten Kurt, Jr.	→	Kurt, M.U.C
Martin U. Kurt, Jr.	→	Kurt, M.U
Martin D. Kurt, Jr.	→	Kurt, M.D.
Emil Kurt	→	Kurt, E.

□

Die Keywords

Eine weitere Datenquelle zur Autorenidentifizierung liefern die Keywords. Mit Hilfe der Keywords lassen sich den einzelnen Autoren Themen zuordnen, wie z.B. Autor a bearbeitet das Gebiet, das durch die Keywords k_i gegeben ist. Abschnitt 5.7 beschreibt ein Verfahren, das es erlaubt, die Publikationen zu klassifizieren. Diese Klassifikationen basieren auf den unterschiedlichen Themengebieten. Dadurch ist es mittels der Keywords möglich, ausgehend von diesen zu den Klassifikationen zu gelangen und im Klassifikationsbaum die Verwandtschaft der Themen zu bestimmen. Zwei Themen sind um so näher verwandt, je geringer der Abstand zwischen den beiden Themen im Klassifikationsbaum ist, oder je geringer der Abstand zum letzten gemeinsamen Vorfahren (ausgehend von der Wurzel) im Baum ist. Je „tiefer“ (weiter von der Wurzel entfernt) der gemeinsame Vorfahre im Baum liegt, desto aussagekräftiger ist dieser berechnete Wert. Ist der gemeinsame Vorfahre die Wurzel, so sind die beiden Themen nicht verwandt, je weiter dieser gemeinsame Vorfahre von der Wurzel entfernt ist, desto mehr passen die Themen zueinander.
Ebenso kann in diesen Wert auch die Entfernung der Themen zum gemeinsamen Vorfahren mit einberechnet werden (die Tiefe des Knotens, der das Thema repräsentiert). Sind beide Themen im Klassifikationsbaum in der selben Tiefe, so definieren diese in etwa ein gleich großes Themengebiet. Je näher ein Knoten der Wurzel ist, desto allgemeiner ist das entsprechende Themengebiet.
Eine weitere Möglichkeit, die Ähnlichkeit zweier Klassifikationen zu bestimmen, ist die Berechnung des kürzesten Pfades zwischen den beiden Themen, wobei die Kanten als ungerichtet betrachtet werden und das Gewicht der Kanten als 1 festgesetzt wird. In diesem Baum kann mittels eines kürzesten Wege Algorithmus (z.B. Dijkstra) der Abstand der Themen bestimmt werden. Der Weg von Thema t_1 zu Thema t_2 ist umso länger, je näher der gemeinsame Vorfahre der Wurzel ist. Ebenso kann die Spezialisierung der Themen mit einberechnet werden, in dem man die Tiefe der Themen bestimmt und diese Tiefe und die Weglänge kombiniert.
Bei beiden Verfahren ist es nötig, die maximale Tiefe d_{max} des Klassifikationsbaumes zu kennen. Sowohl der kürzeste Weg als auch der Abstand zweier Themen ist durch $2 * d_{max}$ beschränkt.

Die LEABib enthält ca. 6560 (8%) Datensätze mit Keywords, das bedeutet, dass die Verwendung der Keywords und damit auch eines Klassifikationsbaumes nur auf diesem kleinen Teil von Daten angewandt werden kann. Um dieses auf alle Daten anwenden zu können, ist es nötig, die Daten mit Keywords anzureichern. In Abschnitt 5.6.1 wurde ein Verfahren vorgestellt das dies bewerkstelligt. Aufgrund der automatischen Anreicherung der Daten ist aber die Qualität der automatisch erzeugten Keywords nicht so hoch, wie die der von Hand erfassten und somit ist auch die Qualität der Klassifizierung dieser Datensätze geringer. Diese Qualitätsunterschiede sollten bei der Berechnung der Ähnlichkeit der Themen mit berücksichtigt werden.

Abbildung 6.3: Klassifikation nach ACM, Version 1998

Das Feld Institution

Dieses Feld gibt an, welche Institution die Forschung, aus der diese Publikation hervorgegangen ist, unterstützt hat. Während einer Forschungsperiode entstehen in der Regel mehrere Publikationen über ein gemeinsames Forschungsgebiet, wobei die Autoren bzw. Autorengruppe über diesen Zeitraum im Allgemeinen konstant bleibt und in jeder Publikation die unterstützende Institution genannt wird. Mit Hilfe dieses Feldinhaltes kann ermittelt werden, ob unterschiedliche Publikationen, die von einem/r Autor/Autorengruppe verfasst wurden, während einer gemeinsamen Forschung entstanden sind. In der LEABib sind in ca. 7000 (8,5%) Datensätzen die Institution angegeben. Dieser Feldinhalt wird mittels einer fest vorgegebenen Werteliste erfasst, so dass die Angaben in diesem Feld von hoher Qualität sind und deshalb auch nur ein String-Vergleich nötig ist, um diese Inhalte zu vergleichen.

Das Feld YEAR

Die Autoren publizieren innerhalb eines bestimmten Zeitfensters, und innerhalb dieses Zeitfensters in mehr oder weniger regelmäßigen Abständen, das heißt ein Autor beginnt im Jahr x mit seiner ersten Publikation und publiziert bis zum Jahr y. Innerhalb der Jahre zwischen x und y veröffentlicht der Autor weitere Artikel. Durch die Anzahl der Artikel und der Größe des Zeitfensters wird die durchschnittliche Dauer zwischen zwei Veröffentlichungen bestimmt. Kennt man die Publikations-Zeitfenster der einzelnen Autoren, so kann dieses bei der Identifizierung der Autoren berücksichtigt werden. Allerdings ist dieses Zeitfenster nicht vorgegeben und muss aus den bestehenden Daten abgeleitet werden.
Das Zeitfenster $[s_i, e_i]$ eines Autors a_i ist bestimmt durch seine Publikationen p_k, wobei $s_i = \min(\text{YEAR})\ \forall p_k$ und $e_i = \max(\text{YEAR})\ \forall p_k$.
Ein Autor a_1 ist auf Basis des Publikationsjahres (Feld YEAR) einem anderen Autor a_2 ähnlich, wenn die Zeitfenster $[s_i, e_i]$ und $[s_j, e_j]$ der Autoren a_i und a_j sich überlappen, oder nahe beieinander liegen ($e_i \approx s_j \vee s_i \approx e_j$).
Bestimmt das Verfahren, dass zwei Autoren a_i und a_j identisch sind, so ändert sich das Zeitfenster der beiden Autoren, da $a_i = a_j$ und somit gilt als neues Zeitfenster $[\min(s_i, s_j), \max(e_i, e_j)]$. Auf Grund des veränderten Zeitfensters ist eine Aktualisierung der bisher berechneten Ähnlichkeitswerte nötig. Dazu sind alle Autoren zu beachten, mit denen bereits ein Vergleich mit a_i oder a_j durchgeführt wurde.

6.3 Algorithmus

Der Algorithmus verwendet ein Cluster-Verfahren um die Anzahl der Autoren-Vergleiche gering zu halten. Die Cluster werden gebildet durch die normalisierten Nachnamen der Autoren. Alle Autoren, deren normalisierter Nachname identisch ist, befinden sich in einem Cluster. Im Fall der LEABib existieren 30449 verschiedene Cluster, wobei der größte Cluster 927 Autoren enthält. Demgegenüber stehen 14776 Cluster, die nur einen einzigen Autor enthalten. Die größten Cluster repräsentieren die Autoren Li (528), Lee (613), Wang (750) und Chen (927). Im Durchschnitt enthält ein Cluster 5.5 Autoren, so dass eine vollständige Suche innerhalb der Cluster schnell durchgeführt werden kann.
Der Ablauf der Autorenidentifizierung ist wie folgt:

Initialisierung *Vergleich der Autorennamen*

Im ersten Schritt erfolgt ein Vergleich auf Basis der Autorennamen. Alle Autoren, deren normalisierter Nachname identisch ist, werden in einem Cluster zusammengefasst.

Diese Cluster werden in eine Queue eingefügt.

Die Schritte 1 bis 6 werden iterativ fortgesetzt, so lange noch Cluster in der Queue sind. Sei i die Anzahl der Elemente in der Queue. Eine Iteration umfasst dann die Abarbeitung der i Cluster in der Queue.

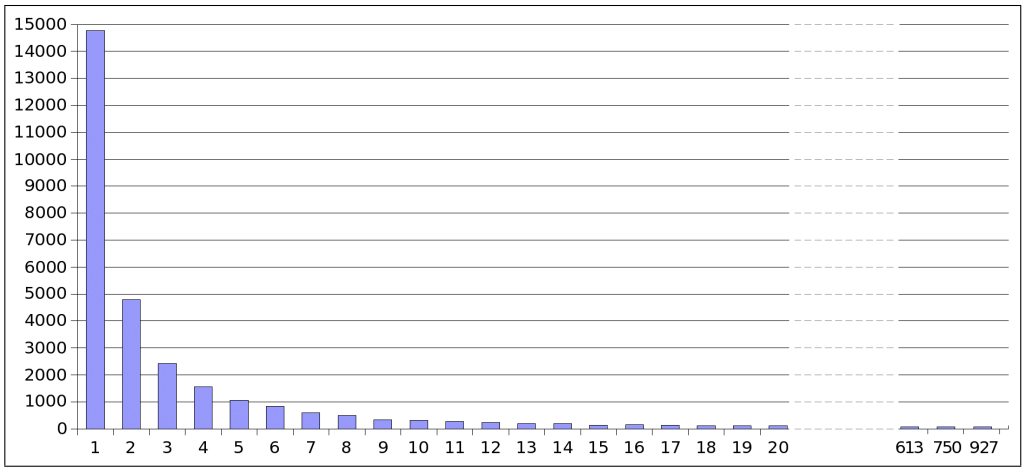


Abbildung 6.4: Clustergrößen

Innerhalb eines Clusters erfolgt eine vollständige Suche nach identischen Autoren. Beim Vergleich zweier Autoren a_1 und a_2 werden dabei folgende Scores bestimmt: Score_N, Score_{AG}, Score_C, Score_I, Score_Y

1. Schritt *Initialen* In einem Cluster befinden sich nur Autoren, die im Nachnamen bereits übereinstimmen. Der Vergleich der Namen kann daher auf die Initialen beschränkt werden.

Seien i_k die Initialen von a_k. Gilt $i_1 = i_2$ oder i_1 ist ein Präfix von i_2 oder i_2 ist ein Präfix von i_1, so sind die beiden Autoren a_1 und a_2 auf Basis der Initialen identisch und liefert als Score_N einen Wert von 1, andernfalls 0.

2. Schritt *Autorengruppen* Sei AG_{a_k} die Menge der Autorengruppen in denen a_k publiziert. Die Menge der Mitautoren ist gegeben durch die Vereinigung der Autoren von AG_{a_1} bzw. AG_{a_2}. Sei $M_{a_1} = \cup AG_{a_1}$ und $M_{a_2} = \bigcup AG_{a_2}$.

$$\text{Score}_{AG} = \begin{cases} 1 & \text{falls } \exists a \text{ mit } a \in M_{a_1} \wedge a \in M_{a_2} \\ 0 & \text{sonst} \end{cases}$$

Gibt es einen Autor, der in beiden Mitautoren-Mengen vorhanden ist, so sind die beiden Autoren a_1 und a_2 auf Basis der Mitautoreneigenschaft als identisch zu betrachten. Ein gemeinsamer Mitautor reicht hierbei schon aus, da im Durchschnitt eine Publikation nur von zwei Autoren verfasst wird (siehe Abschnitt A). Existiert ein Autor in beiden Mengen M_{a_1} und M_{a_2}, der bereits als identisch identifiziert wurde, so ist das Ergebnis des Vergleichs 1, ansonsten müssen die Vergleiche auf Basis der Namen erfolgen. Zwei Autoren i und j sind hierbei identisch, wenn die normalisierten Nachnamen übereinstimmen und für die Initialen der Vergleich, der in Schritt 1 beschrieben ist, 1 liefert.

3. Schritt *Klassifikation* Seien P_{a_i} die Publikationen des Autors a_i und C_P die Klassifikationen der Publikationen in P.

$\text{Score}_C = \max\{d(c_i, c_j) \mid c_i \; inC_{P_{a_1}} \wedge c_j \; inC_{P_{a_2}}\}$, wobei $d(c_i, c_j)$ die Ähnlichkeit der beiden Klassifikationen c_i und c_j angibt.

4. Schritt *Institution* Sei P_{a_i} wie in Schritt 3 definiert und I_P die Institutionen der Publikationen in P.

$$\text{Score}_I = \begin{cases} 1 & \text{falls } \exists x \text{ mit } x \in I_{P_{a_1}} \wedge x \in I_{P_{a_2}} \\ 0 & \text{sonst} \end{cases}$$

5. Schritt *Year* Sei $[s_i, e_i]$ die Zeitspanne, in der die Publikationen der Menge P veröffentlicht wurden. Der Score_T ist um so größer, je näher die beiden Zeitspannen aneinander liegen. Überlappen sie sich oder ist sogar eine Zeitspanne in der anderen enthalten, so ist $\text{Score}_T = 1$.

6. Schritt *Zusammenfassen von Autoren* Nachdem die 5 verschiedenen Scores berechnet wurden, wird entschieden, ob die beiden Autoren a_1 und a_2 identisch sind. Geeignete Regeln definieren die notwendigen Bedingungen, wann die beiden Autoren a_1 und a_2 als gleich zu betrachten sind.

Zeigen die Regeln die Gleichheit von a_1 und a_2, so wird Autor a_2 aus dem Cluster genommen und in die Menge der zu a_1 identischen Autoren eingefügt.

Für Autor a_1 müssen anschließend die entsprechenden Daten aktualisiert werden:

- Hinzufügen der Mitautoren von a_2 zur Liste der Mitautoren von a_1
- Publikationsfenster von a_1 um das Publikationsfenster von a_2 erweitern
- Liste der Institutionen von $_1$ um die Liste der Institutionen von a_2 erweitern
- Klassifikationen von a_2 nach a_1 übernehmen

Zusätzlich wird für jede Menge identischer Autoren ein Autor als Repräsentant gewählt. Als Repräsentant wird der Autor gewählt, der die meisten Initialen besitzt. Ändert sich in einer Menge der Repräsentant, so kann es vorkommen, dass nicht mehr für alle Autoren in dieser Menge die Bedingung auf Basis des Namensvergleichs zutrifft (die Übereinstimmung der Initialen). Aus diesem Grund werden in diesem Fall nochmals die Autoren auf Basis der Namen verglichen und falls ein Autor nicht mehr auf Basis der Initialen zum Repräsentanten passt, so wird dieser Autor aus der Menge entfernt, als Autor in den entsprechenden Cluster eingefügt und der Cluster an die Queue angehängt, falls er nicht bereits in der Queue ist. Durch das Entfernen eines Autors aus der Menge der identischen Autoren ist es notwendig, das Zeitfenster, die Liste der Mitautoren, Institutionen und Klassifikationen erneut aus den noch verbleibenden identischen Autoren zu bestimmen.

Wendet man dieses Verfahren auf die LEABib an, so sind 3 Iterationen nötig. Die 1. Iteration umfasst die Abarbeitung von 30449 Clustern, die 2. Iteration von 19506 und die 3. Iteration

von nur noch 9 Clustern. Dabei benötigte das Verfahren 517 Sekunden und identifizierte 58186 unterschiedliche Autoren.

6.3.1 Regeln zur Identifizierung und Bewertung der einzelnen Felder

Die Klassifikation ist der Feldinhalt mit der geringsten Qualität. Diese Inhalte wurden automatisch aus den Keywords mit Hilfe eines Klassifizierers gewonnen. Darüber hinaus wurden in einer Vorverarbeitung die Daten mit zusätzlichen Keywords, die aus den Titeln extrahiert wurden angereichert, da nur ein kleiner Teil der Daten Keywords enthielt. Durch die automatische Anreicherung der Daten mit Keywords und der darauf aufbauenden Klassifikation kann diese nicht hohen Qualitätsansprüchen genügen. Aus diesem Grund sind die Ergebnisse aus den Klassifikationen nur mit sehr geringem Gewicht in die Bestimmung des Gesamtscores mit einzubeziehen.

Die Ergebnisse aus den Feldern YEAR und INSTITUTION hingegen besitzen eine hohe Qualität, so dass diese Ergebnisse mit hohem Gewicht zum Gesamtscore beitragen.

Die Namen der Autoren wurden in einer Vorverarbeitung normalisiert, so dass unterschiedliche Schreibweisen im Nachnamen fast ausgeschlossen sind, und die Vornamen wurden durch die Initialen ersetzt. Damit ist sichergestellt, dass ein Autor zumindest im Nachnamen in den einzelnen Publikationen übereinstimmt.

Auf Basis der Qualität der Daten kann mit folgenden Regeln sehr zuverlässig die Gleichheit von zwei Autoren a_1 und a_2 bestimmt werden:

Initialen und Mitautoren Sind Score_N und Score_{AG} jeweils 1, so sind die beiden betrachteten Autoren identisch.

Initialen und Institution Sind Score_N und Score_I jeweils 1, so sind die beiden betrachteten Autoren identisch.

Initialen und Veröffentlichungszeitraum Sind Score_N und Score_T jeweils 1, so sind die beiden betrachteten Autoren identisch (Score_T ist 1, wenn sich die Veröffentlichungszeitspannen der beiden Autoren überschneiden oder um maximal 1 Jahr auseinander liegen).

Initialen und Klassifikation Sind Score_N und Score_C jeweils 1, so sind die beiden betrachteten Autoren identisch.

Beispiel:

name: Knuth, Donald E.:::Bender-Knuth/72:::1:::1965-2003
equals: Knuth, Donald E.:::Corless-Jeffrey-Knuth/97:::2:::1997-2003
...
equals: Knuth, D.E.:::Floyd-Knuth/76:::1:::1976-1977
equals: Knuth, Donald Ervin:::Knuth/00:::0:::2000-2000
...
name: Knuth, Donald E.:::Knuth/61:::0:::1961-1961
name: Knuth, D.E.:::Knuth/??:::0
name: Knuth, D.E.:::Knuth/??a:::0

□

Im Beispiel wurde der Autor *Donald E. Knuth* größtenteils richtig erkannt und identifiziert. In den drei Publikationen *Knuth/61*, *Knuth/??* und *Knuth/??a* konnte der Autor allerdings nicht korrekt identifiziert werden, da in diesen Publikationen zum einen kein Mitautor existiert und entweder das Veröffentlichungsjahr fehlt, oder zu weit vom Publikationsfenster *1965-2003* entfernt liegt (4 Jahre, wobei die Zeitdifferenz auf maximal ein Jahr beschränkt ist).

6.3.2 Probleme

Änderungen im Nachnamen

Bei Änderungen im Nachnamen (z.B Heirat), die sich auf den normalisierten Nachnamen auswirken, werden die beiden Autoren in unterschiedliche Cluster einsortiert und bei der clusterbasierten Identifizierung nicht miteinander verglichen und somit auch nicht als identisch erkannt.
Eine Identifizierung dieser Autoren ist leider mit den gegebenen Daten nicht zuverlässig möglich. Welche Autoren sind potentielle Kandidaten zu Autor a, der seinen Nachnamen geändert hat? Mögliche Kandidaten sind die Mitautoren der Mitautoren. Alle Mitautoren von a können nicht mit a identisch sein, aber in den Mitautoren der Mitautoren kann es einen Autor b geben, der zu a identisch ist. Welche Eigenschaften besitzt dieser Autor? Zumindest sollte der Vorname (die Initialen) übereinstimmen. Ein weiteres Kriterium sind die Veröffentlichungszeiträume. Diese sollten entweder nahe beieinander liegen oder sich nur wenig überlappen. Ist ein Veröffentlichungszeitraum im anderen enthalten, so ist b sicher nicht identisch mit a auf Grund einer Änderung des Nachnamens. Versucht man diese Regeln auf die LEABib anzuwenden, so findet man eine sehr große Anzahl von möglichen Umbenennungen von Autoren. Diese Liste liefert zu viele $false - positiv$, so dass die enthaltenen Dubletten nicht gefunden werden können.

Aus dem Autorengraphen lassen sich neben der Mitautorenbeziehung noch weitere Daten ableiten, wie z.B. unterschiedliche Zentralitätsmaße (Grad-, Closeness- und Betweenness-Zentralität) [BE05]. Leider ist es nicht möglich, diese Maße zur Bestimmung identischer Autoren, die ihren Nachnamen geändert haben, heranzuziehen, da eine Änderung des Nachnamens meist nach wenigen Veröffentlichungen geschieht (Amerikanisierung des Namens). Bei Heirat oder anderen Gründen der Namensänderung wäre evtl. eine der Zentralitätsmaße hilfreich, wenn genügend Daten mit „altem“ und „neuem“ Namen vorliegen. Sind die Größen der beiden Datenmengen sehr unterschiedlich, so wirkt sich dies auch entsprechend auf das Zentralitätsmaß aus.

Asiatische Autorennamen

Wie oben erwähnt, umfasst der größte Cluster 927 Autoren mit Nachnamen Chen. Die anderen großen Cluster repräsentieren ebenfalls asiatische Autoren (Li, Lee, Wang). In diesen Clustern befinden sich jeweils unterschiedliche Autoren, deren Initialen identisch sind und somit von der Autorenidentifizierung fälschlicherweise als identisch ($false$-positiv) erkannt werden. Zur Lösung dieses Problems wäre eine Erkennung asiatischer Namen nötig, die dann eine exaktere Überprüfung der Namen durchführt. Fordert man für alle Autoren eine exaktere Übereinstimmung der Vornamen und nicht nur der Initialen, so liefert das Verfahren sehr viele $false$-negativ.

Beispiel:

name: Chen, C.C.:::Alspach-Chen-Heinrich/91:::1:::1989-2005
equals: Chen, Chienhua:::Chen-Agrawal-Burke/94:::0:::1994-1994
equals: Chen, Chiuyuan:::Chen-Chang/96:::0:::1996-1996
equals: Chen, C.C.:::Chen-Chen-Altman/96:::0:::1996-1996
equals: Chen, Chui-Cheng:::Chen-Chen/95:::0:::1995-1995
equals: Chen, Chi-Chang:::Chen-Chen/97:::0:::1997-1997
equals: Chen, Cheng-Chia:::Chen-Lin/93:::0:::1993-1993
equals: Chen, Chen Che:::Chen-Singh-Altman/98:::0:::1998-1998
...

□

In obigem Beispiel wurde in der Publikation Alspach-Chen-Heinrich/91 der Author *Chen, C.C.* (der hier als der 2. Autor angegeben ist) mit den Autoren *Chen, Chienhua*, *Chen, Chiuyuan*, *Chen, C.C.*, ... gleichgesetzt. Betrachtet man diese Liste genauer, so erkennt man allerdings, dass die beiden Autoren *Chen, Chiuyuan* und *Chen, Chui-Cheng* nicht identisch sein können.

6.4 Zusammenfassung

Die Autorenidentifizierung allein auf Basis der Namen liefert i. A. kein brauchbares Ergebnis. Neben rein syntaktischen Methoden (Vergleich des Vor- und Nachnamens) werden weitere Daten zur zuverlässigen Identifizierung herangezogen. Vor allem die Mitautorenbeziehung und die Berücksichtigung der Publikationszeitfenster liefern wertvolle Daten für eine sichere Identifizierung der Autoren. Darüber hinaus unterstützen weitere Daten, die Institutionen und Klassifikationen, die zuverlässige Erkennung. Im Beispiel im Abschnitt 6.3.1 konnte der Autor Donald E. Knuth in der LEABib größtenteils richtig identifiziert werden.

Ein noch offenes Problem stellen zum einen Autoren dar, die ihren Nachnamen ändern, da aus den vorliegenden Daten keine Werte abgeleitet werden können, die eine zuverlässige Identifizierung dieser Autoren ermöglicht. Zum anderen scheitert das Verfahren auch an asiatischen Namen. Hierbei liegt das Problem darin, dass in den einzelnen Clustern, innerhalb derer nach identischen Autoren gesucht wird, viele Autoren existieren, deren Vornamen, als Initialen abgekürzt, identisch sind. Abhilfe könnte hier eine Erkennung der Herkunft eines Autors schaffen, so dass bei asiatischen Namen eine genauere Prüfung der Vornamen erfolgt.

Kapitel 7

Zusammenfassung

Bibliographische Datensammlungen stellen unterschiedliche Anforderungen an die Tools zur Erfassung und Pflege. In dieser Arbeit wurden die Probleme der Dublettenbereinigung und die Identifizierung der Autoren behandelt. Dabei müssen die Verfahren mit den unterschiedlichsten Problemen umgehen.
Die Suche nach Dubletten muss die Anzahl der Vergleiche minimieren, da ein Vergleich zweier Datensätze sehr aufwendig ist. Um zuverlässig alle Dubletten zu finden wäre eine vollständige Suche nötig. Da dies aber in großen Datenbeständen nicht möglich ist, muss ein Kompromiss gefunden werden, der die Anzahl der Vergleiche minimiert, aber mit hoher Zuverlässigkeit alle Dubletten findet.

Auf Grund der unterschiedlichen Fehler (Rechtschreib-, Tipp- und Formfehler) in den zu verarbeitenden Daten können bestehende Algorithmen, wie das Sliding Window oder das Cluster Verfahren, nicht zuverlässig alle Dubletten erkennen. Das Sliding Window* Verfahren, das das Sliding Window Verfahren mit einem Autorengraphen kombiniert und dadurch das Suchfenster entsprechend „vergrößert“, konnte obige Anforderungen erfüllen. Auch durch geschickte Wahl der Clusterfunktion konnten mit Hilfe des Cluster Verfahrens alle Dubletten in Testdaten erkannt werden, wobei das Cluster Verfahren die bessere Laufzeit bietet, aber bei bestimmten Fehlern in den Daten evtl. nicht alle Dubletten findet. Robuster gegenüber Erfassungsfehler, aber auch entsprechend langsamer (Faktor 14), arbeitet das Sliding Window* Verfahren. Hierbei ist es fast ausgeschlossen, dass durch Erfassungsfehler Dubletten nicht erkannt werden.
Da ständig neue Daten erfasst und in der LEABib veröffentlicht werden, waren Erweiterungen in den Verfahren nötig, um auch das Problem der inkrementellen Deduplizierung zu lösen. Mit Hilfe von geeigneten Zusatzinformationen, die aus dem vorausgegangenem Ablauf gewonnen wurden, war es möglich die bestehenden Verfahren entsprechend abzuändern.

Neben geeigneten Verfahren zur Suche nach den Dubletten sind auch Methoden und Regeln nötig, die entscheiden, ob zwei Datensätze identisch sind. Es wurde hier, ausgehend von einem naiven Ansatz, ein Algorithmus entwickelt, der sehr zuverlässig entscheidet, ob die beiden zu vergleichenden Daten identisch sind. Dabei verwendet das Verfahren unter-

schiedliche Heuristiken und bei den Stringvergleichen kommt die Edit-Distanz zum Einsatz. Um die Berechnung der Edit-Distanz zu beschleunigen, erfolgt zuerst eine untere Abschätzung der Edit-Distanz und falls diese berechnet wird, wird diese mit einem optimierten Algorithmus bestimmt. Durch diese beiden Optimierungen wurde die Berechnung um einen Faktor 2 beschleunigt.

Die Autorenidentifizierung erfolgt mit einem iterativen, clusterbasierten Verfahren. Dabei wird zum einem die Mitautorenschaft verwendet, die aus dem Publikationsgraphen gewonnen wird, und zum anderen mehrere Heuristiken, um eine zuverlässige Erkennung zu gewährleisten. Zu diesen Heuristiken zählen u.a. die Normalisierung der Daten, Bildung der Initialen der Vornamen, Berücksichtigung der Publikationszeiträume. Diese Heuristiken funktionieren sehr gut mit europäischen und englischen/amerikanischen Namen, bei asiatischen Namen scheitert das Verfahren leider teilweise. Abhilfe könnte hier eine Erkennung der Herkunft der Namen schaffen, wobei für die unterschiedlichen Regionen verschiedene Heuristiken zur Identifizierung verwendet werden.

Anhang A

Statistik LEABib

Diese Statistik gibt die aktuellen Zahlen der LEABib wieder.

Anzahl Daten:	82364
Anzahl Autoren:	168876
Max. Anzahl Autoren in Datensatz:	62
Durchschnittliche Anzahl Autoren pro Datensatz:	2.05
Anzahl Daten mit Keywords:	6561 (7.97%)
Anzahl Keywords:	30369
Max. Anzahl Keywords:	28
Durchschnittliche Anzahl Keywords:	4.63

Tabelle A.1: Daten der LEABib

CITKEY	82364	100,00 %
AUTHOR	82312	99,94 %
TITLE	81756	99,26 %
BOOKTITLE	26816	32,56 %
JOURNAL	52866	64,19 %
SERIES	12852	15,60 %
VOLUME	65784	79,87 %
NUMBER	40516	49,19 %
YEAR	82008	99,57 %
PAGES	79164	96,11 %
EDITOR	26342	31,98 %
KEYWORDS	6561	7,97 %
ABSTRACT	6345	7,70 %
INSTITUTION	7034	8,54 %
ORGANIZATION	8595	10,44 %
PUBLISHER	78029	94,74 %
URL	25129	30,51 %
TYPE	5788	7,03 %
NOTE	969	1,18 %
PCOMMENT	859	1,04 %

Tabelle A.2: Verteilung der Felder

A.1 Verteilung der Buchstaben

Die folgende Tabelle zeigt die Buchstabenverteilungen für die LEABib, deutsche und englische Texte.

Zeichen	LEABib	Englisch	Deutsch
␣	9.2%	18.2%	15.1%
e	7.4%	9.0%	12.7%
a	6.3%	6.5%	4.6%
n	6.1%	5.5%	7.8%
o	5.8%	5.9%	2.3%
i	5.4%	5.9%	6.1%
r	5.3%	5.0%	5.7%
s	4.9%	5.0%	5.0%
t	4.7%	7.0%	5.2 %
l	3.4%	3.2%	3.0%
c	3.0%	2.4%	2.4%
⋮			

Tabelle A.3: Verteilung der Buchstaben

A.2 Laufzeiten

Die unten stehende Tabelle zeigt die benötigten Laufzeiten zur Bestimmung der Dubletten in einer Datenmenge der Größe 17397 und den Einfluss der verschiedenen Abschätzungen der Edit-Distanz.

Optimierung	Laufzeit	Steigerung der Geschwindigkeit
keine	49243	
Vergleich der Stringlängen	34432	1.4
Fingerprint über Σ	26465	1.8
Fingerprint über $\Sigma \wedge$ Vergleich der Stringlängen	23722	2.0
Fingerprint über S	34672	1.4
Fingerprint über $S \wedge$ Vergleich der Stringlängen	30715	1.6

Tabelle A.4: Laufzeiten

mit $S = \{$␣, e, a, n, o, i, r, s, t$\}$.

A.3 Verteilung Anzahl Autoren

Die folgende Tabelle gibt die Verteilung der Anzahl der Autoren bezogen auf die Publikationen an. Die erste Spalte gibt die Anzahl der Autoren an, die zweite, wie viele Publikationen es mit dieser Anzahl von Autoren gibt.

0	52	0,0308 %
1	30307	17,9463 %
2	30363	17,9795 %
3	14025	8,3049 %
4	4994	2,9572 %
5	1533	0,9078 %
6	559	0,3310 %
7	237	0,1403 %
8	117	0,0693 %
9	50	0,0296 %
10	33	0,0195 %
11	33	0,0195 %
12	17	0,0101 %
13	11	0,0065 %
14	5	0,0030 %

15	6	0,0036 %
16	3	0,0018 %
17	2	0,0012 %
18	2	0,0012 %
19	3	0,0018 %
20	4	0,0024 %
21	1	0,0006 %
22	1	0,0006 %
23	1	0,0006 %
25	1	0,0006 %
28	1	0,0006 %
44	1	0,0006 %
48	1	0,0006 %
62	1	0,0006 %

Tabelle A.5: Verteilung der Autoren

A.4 Verteilung Anzahl Keywords

Die folgende Tabelle gibt die Verteilung der Anzahl der Keywords bezogen auf die Publikationen an. Die erste Spalte gibt die Anzahl der Keywords an, die zweite, wie viele Publikationen es mit dieser Anzahl von Keywords gibt.

0	75803	92,0341 %	12	29	0,0352 %
1	97	0,1178 %	13	18	0,0219 %
2	426	0,5172 %	14	7	0,0085 %
3	1492	1,8115 %	15	8	0,0097 %
4	1688	2,0494 %	16	5	0,0061 %
5	1253	1,5213 %	17	2	0,0024 %
6	709	0,8608 %	18	1	0,0012 %
7	370	0,4492 %	19	2	0,0024 %
8	191	0,2319 %	20	1	0,0012 %
9	133	0,1615 %	21	1	0,0012 %
10	89	0,1081 %	22	1	0,0012 %
11	37	0,0449 %	28	1	0,0012 %

Tabelle A.6: Verteilung der Keywords

Anhang B

Gewichtung der Felder

Feld k	$c(k)$
TYPE	1.0
AUTHOR	0.8
TITLE	0.8
BOOKTITLE	0.5
JOURNAL	1.0
PAGES	1.0
YEAR	1.0
NUMBER	0.8
VOLUME	0.8
SERIES	1.0
KEYWORDS	0.1
ABSTRACT	0.0
EDITOR	0.8
PUBLISHER	0.5
INSTITUTION	0.5
ORGANIZATION	0.5
URL	0.1

Tabelle B.1: Gewichtung der Felder

Die Art der Publikation (TYPE) trennt die Daten nur teilweise, nur bestimmte Kombinationen der Typen trennen zwei Datensätze. Die folgende Auflistung zeigt den Zusammenhang zwischen den einzelnen Typen und deren Trenn-Eigenschaften.

BibTEXdefiniert folgende Publikationsarten: *article, book, booklet, conference, inbook, incollection, inproceedings, manual, mastersthesis, misc, phdthesis, proceedings, techreport* und

unpublished.
Die Typen *misc* und *unpublished* passen zu allen anderen Typen. *book, booklet* und *incollection* werden als gleich betrachtet. Ebenso *conference, inproceedings* und *proceedings*. Alle anderen Typen werden als verschieden betrachtet.

Anhang C

Keywords

In der folgenden Tabelle ist der Einfluss der Stoppwortliste, des Stemmings und des Cut-Offs auf die Anzahl der Keywords aufgelistet.

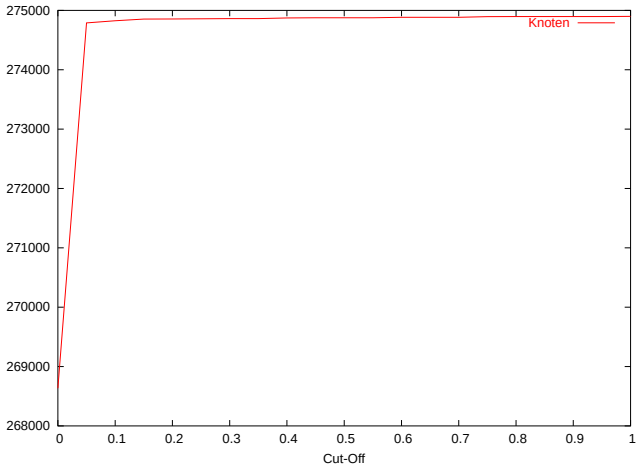


Abbildung C.1: Zusammenhang Cut-Off und Knoten

	Anzahl Keywords	Anzahl unterschiedlicher Keywords	ZHK	Knoten	Kanten
¬Stoppwortliste ∧¬Stemming	510940	14968	531	275873	769541
Stoppwortliste ∧¬Stemming	398793	14933	540	275839	657402
¬Stoppwortliste ∧ Stemming	514514	14769	533	276760	773114
Stoppwortliste ∧ Stemming	405856	14745	540	276736	664464
Cut-Off (10%) ∧ Stoppwortliste ∧ Stemming	359777	14738	562	276731	618390

Cut-Off	Knoten	Kanten	ZHK	Größe
0.00	271615	510153	902	268636
0.05	276726	601226	571	274790
0.10	276731	618390	562	274826
0.15	276734	631087	554	274853
0.20	276735	637774	553	274856
0.25	276735	645014	552	274858
0.30	276736	647185	551	274862
0.35	276736	647829	551	274862
0.40	276736	649571	548	274872
0.45	276736	650748	547	274875
0.50	276736	650748	547	274875
0.55	276736	650748	547	274875
0.60	276736	652010	544	274883
0.65	276736	652010	544	274883
0.70	276736	652010	544	274883
0.75	276736	655054	542	274894
0.80	276736	660698	541	274897
0.85	276736	660698	541	274897
0.90	276736	660698	541	274897
0.95	276736	660698	541	274897
1.00	276736	664464	540	274899

Tabelle C.1: Einfluss der Stoppwortliste, des Stemmings und des Cut-Offs

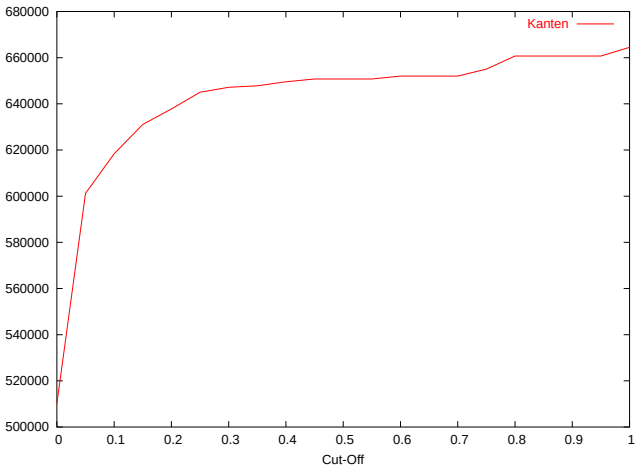


Abbildung C.2: Zusammenhang Cut-Off und Kanten

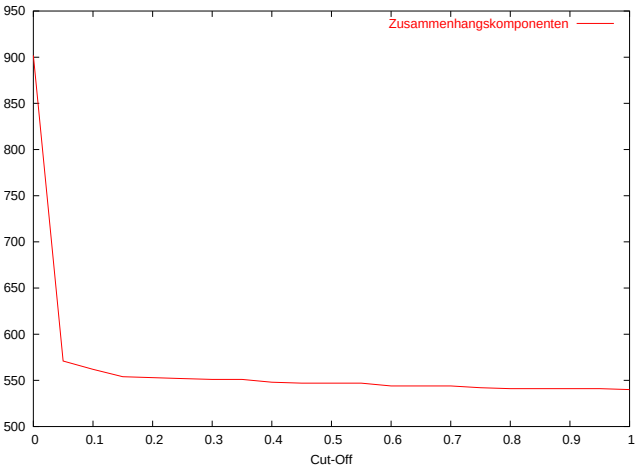


Abbildung C.3: Zusammenhang Cut-Off und Anzahl der Zusammenhangskomponenten

LEABib, um zusätzliche Keywords angereichert		LEABib			
Größe	Anzahl	Größe	Anzahl	Größe	Anzahl
111177	1	72800	1	26	1
17	2	3227	1	23	1
11	1	1604	1	22	3
8	1	977	1	20	1
7	6	838	1	19	3
6	4	742	1	17	4
5	13	400	1	16	2
4	17	269	1	15	4
3	27	247	1	14	1
2	83	182	1	13	2
1	406	59	1	12	2
		57	1	11	4
		54	1	10	5
		53	1	9	1
		52	1	8	6
		50	1	7	8
		44	1	6	5
		37	1	5	23
		35	1	4	27
		32	1	3	52
		31	1	2	165
		29	1	1	15987

Tabelle C.2: Anzahl und Größe der Zusammenhangskomponenten

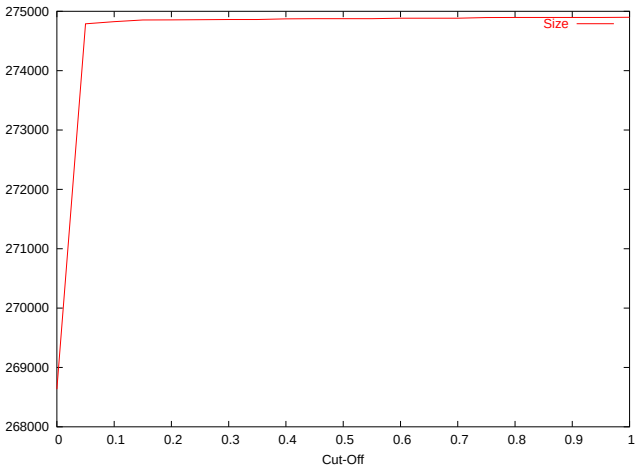


Abbildung C.4: Zusammenhang Cut-Off und Größe der maximalen Komponente

C.1 Stoppwortliste

A	AT	EITHER	IS	OUR	THEREFORE	WHEN
ABOUT	BE	FOR	IT	SEE	THESE	WHERE
AFTER	BECAUSE	FROM	ITS	SEEN	THEY	WHETHER
ALL	BEEN	FURTHER	MAY	SHOULD	THIS	WHICH
ALREADY	BETWEEN	HAD	MORE	SINCE	THOSE	WHILE
ALSO	BOTH	HAS	MOREOVER	SUCH	THOUGH	WHOSE
ALTHOUGH	BUT	HAVE	MOST	THAN	THROUGH	WILL
ALWAYS	BY	HAVING	MUST	THAT	THUS	WITH
AMONG	COULD	HERE	NO	THE	TO	WITHIN
AN	DO	HOWEVER	OF	THEIR	WAS	WOULD
ANY	DOES	IF	ON	THEM	WE	
ARE	DURING	IN	ONLY	THEN	WERE	
AS	EACH	INTO	OTHER	THERE	WHAT	

(aus http://stneasy.cas.org/html/english/helps/2search/2B4stopw.htm)

Literaturverzeichnis

[Bar95] Burkhard Bartsch. Diplomarbeit: Approximatives Patternmatching, 1995.

[BD83] Dina Bitton and David J. DeWitt. Duplicate record elimination in large data files. In *Proceedings of the ACM Transactions on Database Systems, TODS'1983*, volume 8, pages 255 – 265, New York, USA, 1983. ACM Press.

[BE05] Ulrik Brandes and Thomas Erlebach. *Network Analysis*, volume 3418 of *LNCS*. Springer, 2005.

[Ber02] Pavel Berkhin. Survey Of Clustering Data Mining Techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[BM02] Mikhail Bilenko and Raymound J. Mooney. Learning to combine trained distance metrics for duplicate detection in databases. Techreport, University of Texas, Austin, Texas, 2002.

[Bro05] Felix Brosius. *SPSS-Programmierung*. Mitp-Verlag, 2005.

[Buh01] Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[CDEV02] M. Cochinwala, S. Dahl, A.K. Elmagarmid, and V.S. Verykios. Record matching: Past, present and future, 2002.

[Cha97] Carole E. Chaski. Who wrote it? *National Institute of Justice Journal*, 233, 1997.

[CM91] D. Campbell and T. McNeill. Finding a majority when sorting is not available. *The Computer Journal*, 34(2):186, 1991.

[Dei04] Anika Deinert. Die Häufigkeitsverteilung der Buchstaben in deutschen Texten zur kypotanalytischen Anwendung, 2004.

[dFS04] Statistisches Landesamt des Freistaates Sachsen. Häufigkeit der Verteilung von Namensanfängen, 2004.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[Far96] Jill M. Farringdon. How to be a literary Detective: Authorship attribution, 1996.

[FMS93] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. The complexity of finding replicas using equality tests. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Proceedings of the 18th Mathematical Foundations of Computer Science, MFCS'93*, volume 711 of *Lecture Notes in Computer Science*, pages 463–472. Springer, 1993.

[GFS+01a] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Christian-Augustin Saita. Improving data cleaning quality using a data lineage facility, 2001.

[GFS+01b] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian Saita. Declarative data cleaning: Language, mode and algorithms, 2001.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *The VLDB Journal*, pages 518–529, 1999.

[Gus97] Dan Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, Cambridge, 1997.

[Ham50] Richard W. Hamming. Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[Han03] Matthias Hanitzsch. SEP - Dokumentation: Online-Referenzengenerierung, 2003.

[HEG06] Jian Huang, Seyda Ertekin, and C. Lee Giles. Efficient name disambiguation for large-scale databases. In *10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 536–544, 2006.

[HLL+03] Lutz Horn, Michael Ley, Peter Luksch, Jörg Maas, Ernst W. Mayr, Andreas Oberweis, Paul Ortyl, Stefan Pfingstl, Enzo Rossi, Felix Rüssel, Ute Rusnak, Daniel Sommer, Wolffried Stucky, Roland Vollmar, and Marco von Mevius. Konzeption und Betrieb eines Kompetenz- und Dienstleistungsnetzes für die Informatik. In *GI Jahrestagung*, pages 132–147, 2003.

[HM91] Xiaoqiu Huang and Webb Miller. A time-efficient linear-space local similarity algorithm. *Adv. Appl. Math.*, 12(3):337–357, 1991.

[HS95] Mauricio A. Hernandez and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In *SIGMOD Conference*, pages 127–138, 1995.

[HS98] Mauricio A. Hernández and Salvatore J. Stolfo. Real world data is dirty: Data cleansing and the Merge/Purge problem. *Data Mining and Knowledge Discovery*, 2:9–37, 1998.

[HT84] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[Ini] Dublin Core Metadata Initiative. Dublic Core.

[Jon] David M. Jones. ACM Computing Classification System. WWW.

[Kir] Clemens Kirchmair. *LEABib Dokumentation.*

[Kro93] R. Krovetz. Viewing Morphology as an Inference Process. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–203, 1993.

[Kus00] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[KWD97] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper Induction for Information Extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[LLL00] Mong Li Lee, Tok Wang Ling, and Wai Lup Low. IntelliClean: A knowledge-based intelligent data cleaner. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'2000*, pages 290–294, 2000.

[ME96] Alvaro E. Monge and Charles P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'96*, pages 267–270, 1996.

[ME97a] Alvaro E. Monge and Charles Elkan. An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In *Research Issues on Data Mining and Knowledge Discovery*, pages 0–, 1997.

[ME97b] Alvaro E. Monge and Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Workshop on Research Issues on Data Mining and Knowledge Discovery, DMKD'97*, 1997.

[MMK01] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[Mon97] Alvaro Edmundo Monge. Adaptive Detection of Approximately Duplicate Database Records and the Database Integration Approach to Information Discovery, 1997.

[Mon00] Alvaro E. Monge. An Adaptive and Efficient Algorithm for Detecting Approximately Duplicate Database Records, 2000.

[Mut04] Peter Mutschke. Autorennetzwerke: Netzwerkanalyse als Mehrwert für Informationssysteme. In *Proceedings des 9. internationalen Symposiums für Informationswissenschaft, ISI'2004 (Chur, 6.-8. Oktober 2004*, pages 141–162. UVK Verlagsgesellschaft, 2004.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[OP04] Paul Ortyl and Stefan Pfingstl. Extrahierung bibliographischer Daten aus dem Internet. In *GI Jahrestagung*, pages 203–207, 2004.

[Pat88] Oren Patashnik. BIBTEXing. WWW, 1988.

[Por97] M.F. Porter. An algorithm for suffix stripping. *Readings in Information Retrieval*, pages 313–317, 1997.

[PW01] Edward H. Porter and William E. Winkler. Approxiamte string comparsion and its effect on an advanced record linkage system. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *Proceedings of the 27th VLDB Conference*, pages 381–390. Morgan Kaufmann, 2001.

[Res01] Thomson ResearchSoft. RIS Format Specifications, 2001.

[RH01] Vijayshankar Raman and Joseph M. Hellerstein. Potter's Wheel: An interactive data cleaning system. In *Proceedings of 27th International Conference on Very Large Data Bases, VLDB'2001*, 2001.

[RRP80] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. New models in probabilistic information retrieval, 1980.

[SS01] B. Schölkopf and A.J. Smola. *Learning with kernels support vector machines, regularization, optimization and beyond.* MIT Press, 2001.

[STN] STNEasy. STNEasy: Stop Words.

[Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64, 1985.

[VEH99] Vassilios S. Verykios, Ahmed K. Elmagarmid, and ELias N. Houstis. Record matching to improve data quality, 1999.

[VVSK01] Panos Vassiliadis, Zografoula Vagena, Spiros Skiadopoulos, and Nikos Karayannidis. ARKTOS: A tool for data cleaning and transformation in data warehouse enviroments, 2001.

[Wik06a] Wikipedia. List of frequently misused English words, 2006.

[Wik06b] Wikipedia. Rechtschreibfehler, 2006.

Stichwortverzeichnis