

# Indizierungstechniken im Information Retrieval

Steffen Lang

Seminar Spezifikations- und Selektionsmethoden für Daten und Dienste  
Wintersemester 2005/06

Universität Karlsruhe (TH)  
IPD, Lehrstuhl Prof. Böhm

## 1 Einleitung

Im Information Retrieval werden Indizierungstechniken dazu benutzt, relevante Dokumente innerhalb einer Dokumentensammlung entsprechend einer Anfrage, die ein gewisses Informationsbedürfnis widerspiegelt, effizient zu ermitteln. Dabei wurden die ersten Techniken schon vor langer Zeit entwickelt, z. B. der lexikographisch sortierte Index in einem Buch, welcher zu den wichtigen Begriffen sortierte Listen mit deren Vorkommen bereithält. Dadurch wird die gezielte Suche nach bestimmten Inhalten erheblich beschleunigt. Im modernen Information Retrieval entspricht diese Struktur den sogenannten Invertierten Listen. Diese sowie die Signatur-Dateien, die Suffix-Arrays und das Latent Semantic Indexing werden im folgenden Kapitel näher vorgestellt. Dabei wird versucht, deren Eignung für verschiedene Retrieval-Modelle, die jeweilige Dauer einer Anfragebearbeitung und die Größe des Index (in Abhängigkeit der Größe  $n$  der zugrundeliegenden Dokumentensammlung) zu verdeutlichen.

## 2 Indizierungstechniken

### 2.1 Invertierte Listen

Unter einer Invertierten Liste versteht man die Zuordnung einer Liste zu jedem *Indexterm*, welche aus den Dokument-IDs der ihn enthaltenden Dokumente besteht. Ein Indexterm ist hierbei ein Term, welcher als charakteristisch für den Dokumentinhalt betrachtet wird, und deshalb zur Indizierung verwendet wird. Die Menge der alphabetisch sortierten Indexterme bezeichnet man als *Vokabular*, die Menge der Listen als die *Vorkommen*. Diese können neben den IDs noch Gewichte enthalten, welche bei manchen Retrieval-Modellen benötigt werden. Ein einfaches Beispiel aus 3 Sätzen, von denen jeder einem Dokument entspricht, und der zugehörigen Invertierten Liste befindet sich in Abbildung 1, wobei die fettgedruckten Worte Indexterme sind. Weitere grundlegende Aspekte der Invertierten Listen werden in [HBYFL92] beschrieben.

**Ablauf einer Anfrage** Die Terme einer Anfrage müssen alle als einzelne Anfragen behandelt werden, und dann deren Ergebnisse entsprechend der Anfrage kombiniert werden. Dies liegt daran, dass nur die Invertierten Listen einzelner Indexterme durch einen Zugriff auf das Vokabular ermittelt werden können. Die Kombination der Ergebnisse erfolgt beim Booleschen Retrieval durch einfache Mengenoperationen, bei anderen Retrieval-Modellen müssen noch weitere Operationen ausgeführt werden, wie beispielsweise das Berechnen eines Rankings.

Die Anfrage für einen Term läuft folgendermaßen ab: Zuerst wird der Term im Vokabular gesucht, und daraufhin mittels des gespeicherten Pointers die zugehörige Liste aus den Vorkommen geladen.

Der Aufwand der Anfragebearbeitung hängt somit von der Datenstruktur des Vokabulars, der Länge der Listen und dem Typ der Anfrage ab. Laut [BYRN99] kann gezeigt werden, dass bei sinnvoller Selektivität die Zeitkomplexität selbst für komplizierte Anfragen in  $O(n^{0.8})$  liegt.



dann bietet sich auch noch eine in [BCC94] beschriebene Implementierungsmethode an.

**Kompression** Die erzielbaren Kompressionsraten bei Invertierten Listen sind sehr hoch. Dies liegt vor allem daran, dass die Listen mit den Dokument-IDs aufsteigend sortiert sind, so dass man statt den IDs einfach die Differenz zur nächsten ID speichern kann. Dabei verwendet man Kodierungen, welche für kleine Differenzen nur wenige Bit benötigen, weil diese bei Termen mit großen Häufigkeiten, und damit insgesamt häufiger auftreten. Da die Listen normalerweise sequentiell durchlaufen werden, beeinträchtigt diese Art der Kompression die Anfragebearbeitung kaum.

## 2.2 Signatur-Dateien

Für die Indizierung mit Signatur-Dateien werden die Dokumente in ungefähr gleich große Blöcke zerlegt, welche dann jeweils mittels einer Hashfunktion  $h$  auf Signaturen der Größe  $B$  abgebildet werden. Dies geschieht dadurch, dass die Hashwerte der Indexterme eines Blocks bitweise mit ODER verknüpft werden. Die Signaturen aller Blöcke ergeben dann die Signatur-Datei eines Dokuments. In den Abbildungen 2 - 4 sind ein Dokument mit vier Blöcken, eine Hashfunktion für die fettgedruckten Indexterme, und die zugehörige Signatur-Datei abgebildet.  $B$  hat dabei zur besseren Übersichtlichkeit den kleinen Wert 3. Eine weitergehende Beschreibung und Analyse von Signatur-Dateien befindet sich in [FC84].

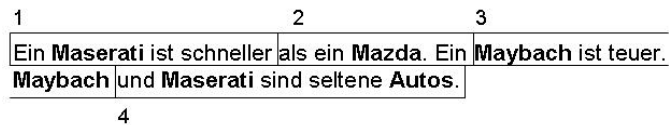


Abbildung 2. Beispieldokument mit vier Blöcken

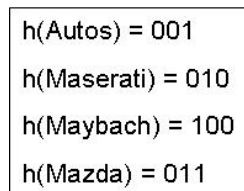


Abbildung 3. Hashfunktion

010 011 100 011

Abbildung 4. Signatur-Datei

**Ablauf einer Anfrage** Um boolesche ODER-Anfragen zu beantworten, werden ähnlich wie bei Invertierten Listen Mengenoperationen angewendet. Dazu werden die Mengen der gefundenen Blöcke vereinigt. Für UND-Anfragen gibt es jedoch ein anderes Vorgehen. Dazu wird für jeden Term der Anfrage der Hashwert berechnet, und diese Werte daraufhin durch bitweises ODER verknüpft, so dass man eine Signatur der Anfrage erhält. In Abbildung 5 ist hierfür ein Beispiel, welches die Hashfunktion aus Abbildung 3 verwendet. Diese Anfragesignatur wird daraufhin mittels einer bitweisen UND-Verknüpfung mit den jeweiligen Blocksignaturen verglichen. Falls das Ergebnis der Verknüpfung der Anfragesignatur entspricht, so handelt es sich um einen potentiellen Treffer. Für die Anfrage aus Abbildung 5 ist dies bei den Blöcken 2 und 4 der Fall, obwohl nur Block 4 der Anfrage entspricht. Bei dem potentiellen Treffer für Block 2 handelt es sich deshalb um einen sogenannten *False Positive*. Diese kann man entweder hinnehmen, oder durch eine zusätzliche, direkte Suche in den Texten der betroffenen Blöcke ausschließen.

Ein weiteres Problem bei Signatur-Dateien stellen die Blockgrenzen dar, da sie dazu führen können, dass nahestehende Terme nicht als solche erkannt werden. Zum Beispiel würde eine Suche nach einem Satz, der „Maybach“ und „Maserati“ beinhaltet, für die Blockeinteilung aus Abbildung 2 keinen Treffer liefern, obwohl so ein Satz vorkommt. Um dieses Problem zu lösen, lässt man die Blöcke soweit überlappen, dass die Nähe von Termen bis zu einer gewünschten Grenze noch erkannt werden kann.



Abbildung 5. Anfrage mit Signatur

Natürlich sind nicht nur reine ODER- bzw. UND-Anfragen möglich. Eine Anfrage, die nur aus einem Term besteht, wird wie eine UND-Anfrage behandelt, wobei der Hashwert des Terms direkt die Anfragesignatur ergibt. Gemischte Anfragen lassen sich z. B. dadurch bewältigen, dass man die Anfrage in eine Disjunktive Normalform bringt, woraufhin man zuerst die UND-Anfragen bearbeitet und die Resultate dann vereinigt.

Die Anfragedauer hängt somit hauptsächlich von der Anzahl der Blöcke ab, und liegt insgesamt in  $O(n)$ . Dabei hat die Komplexität aber eine kleine Konstante, da durch geeignete Wahl der Signaturgröße  $B$  die Bitoperationen sehr schnell ausgeführt werden können. Deshalb sind Signatur-Dateien vor allem bei kleinen Dokumentensammlungen relativ schnell.

**Größe** Die Größe der Signatur-Dateien hängt von der Signaturgröße und linear von der Größe der Dokumentenkollektion ab, und liegt somit in  $O(n)$ . Bei der Wahl der Signaturgröße muss man einen Kompromiss zwischen Indexgröße, Anzahl der False Positives und Genauigkeit der Treffer eingehen.

**Retrieval-Modelle** Signatur-Dateien sind in der hier vorgestellten Form nur für das Boolesche Retrieval geeignet, da keinerlei Gewichte oder Häufigkeiten gespeichert werden, welche von anderen Modellen benötigt werden.

**Kompression** Die Komprimierung von Signatur-Dateien ist gut möglich, da nach [BYRN99] nur wenige Bits in den Dateien gesetzt sind. Daher lassen sie sich beispielsweise mit einer Lauflängenkodierung effektiv komprimieren.

### 2.3 Suffix-Arrays

Bei Suffix-Arrays handelt es sich um eine kompakte Speicherform für Suffix-Trees, welche hier nur kurz erläutert werden sollen:

Im Suffix-Tree einer Dokumentenkollektion werden alle in ihr enthaltenen Suffixe gespeichert. Dabei stehen die Kanten des Baumes für Zeichen, und die Blätter für Verweise auf die Vorkommen. Die Gesamtheit aller Pfade von der Wurzel zu den Blättern ergibt alle gespeicherten Suffixe.

In Abbildung 6 ist ein Suffix-Tree für den String „ababc“ abgebildet, wobei die Kanten lexikographisch sortiert sind.

Ein Suffix-Array ist ein Array mit den Verweisen aus den Blättern des Suffix-Trees in lexikographischer Reihenfolge. In Abbildung 7 ist das Suffix-Array für den Baum aus Abbildung 6 dargestellt. Suffix-Arrays wurden 1990 in [MM90] vorgestellt. Unabhängig davon wurden sie schon 1987 in [Gon87] unter den Namen „PAT-Arrays“ eingeführt.

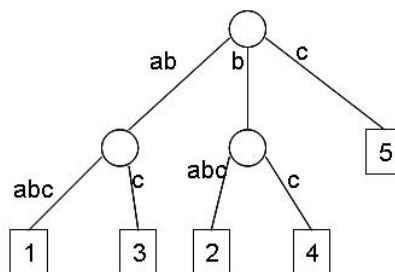


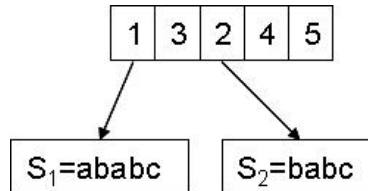
Abbildung 6. Suffix-Tree für „ababc“



**Abbildung 7.** Suffix-Array für „ababc“

**Konstruktion** Ein Suffix-Array lässt sich aus einem Suffix-Tree, bei dem die Kanten lexikographisch sortiert sind, durch eine einfache Tiefensuche erzeugen, wobei der Inhalt jedes Blattes, sobald es erreicht ist, in das Array geschrieben wird. Ist der Suffix-Tree nicht sortiert, so muss bei der Tiefensuche explizit darauf geachtet werden, dass an der richtigen Kante hinabgestiegen wird. Eine weitere Möglichkeit ein Suffix-Array zu erzeugen kommt komplett ohne den Suffix-Tree aus. Dazu werden zuerst alle zu indizierenden Suffixe ermittelt, und diese dann sortiert. Daraufhin werden die Suffixe der Reihe nach durchgegangen und ein jeweiliger Verweis im Array gespeichert.

**Ablauf einer Anfrage** Um eine Anfrage mittels eines Suffix-Arrays auszuwerten, wird diese als ein String interpretiert. Daraufhin werden durch binäres Suchen im Array die Suffixe  $S_1$  und  $S_2$  ermittelt, so dass  $S_1$  der erste Suffix ist, welcher der Anfrage entspricht, und  $S_2$  der Nachfolger des letzten. Für die Vergleiche bei der binären Suche muss jeweils auf die Dokumente zugegriffen werden, da im Array nur Verweise stehen. Das Ergebnis einer Anfrage sind dann alle Verweise, die sich zwischen den Verweisen auf  $S_1$  (inklusive) und  $S_2$  (exklusive) befinden. In Abbildung 8 wird der Vorgang an einem Beispiel verdeutlicht.



**Abbildung 8.** Suche nach „ab“ in „ababc“

Um Anfragen, die aus mehreren Termen bestehen, zu beantworten, können die Ergebnismengen wieder kombiniert werden. Die Stärke der Suffix-Arrays liegt jedoch im Auffinden von Mustern in nicht-textuellen Daten. So eignen sie sich beispielsweise hervorragend für die Suche von wiederkehrenden Mustern in Gensequenzen.

Der Aufwand einer Anfrage liegt dank der binären Suche in  $O(\log n)$ , wobei aber durch die vielen wahlfreien Zugriffe auf die Dokumente bei den Vergleichen ein hoher Hintergrundaufwand entsteht, so dass in der Praxis Invertierte Listen schneller sind.

**Größe** Die Größe eines Suffix-Arrays hängt nur von der Anzahl der Suffixe ab, da für jeden ein Verweis gespeichert wird. Somit liegt die Platzkomplexität in  $O(n)$ .

**Retrieval-Modelle** Suffix-Arrays sind in der Form, wie sie hier beschrieben wurden, nur für das Boolesche Retrieval geeignet. Für andere Modelle müssten zusätzliche Daten, wie etwa Gewichte, im Array gespeichert werden, oder aber besser auf Suffix-Trees zurückgegriffen werden, da diese mehr Möglichkeiten bieten, zusätzliche Informationen in einzelnen Knoten zu speichern.

**Kompression** Suffix-Arrays sind sehr schlecht zu komprimieren, da sie aus einer zufällig verteilten Permutation von Verweisen auf die Dokumentensammlung bestehen.

## 2.4 Latent Semantic Indexing

Beim Latent Semantic Indexing (LSI) handelt es sich nicht um eine reine Indizierungstechnik, sondern auch um ein eigenes Retrieval-Modell, welches ebenfalls unter dem Namen Latent Semantic Analysis bekannt ist. Darüberhinaus lässt es sich auch zur Clusterung von Dokumenten einsetzen. Grundlage des LSI ist die *Term-Dokument-Matrix*, welche jedem Term ein Gewicht in jedem Dokument zuordnet. In Abbildung 9 ist eine solche Matrix für sieben Dokumente und fünf Terme dargestellt, wobei die Werte für dieses Beispiel von [Böh05] übernommen wurden. Da solche Matrizen im Allgemeinen aufgrund vieler Terme und vieler Dokumente hochdimensional sind, versucht man diese durch eine Singulärwertzerlegung und anschließende Dimensionsreduktion zu verkleinern. Hier soll nicht näher auf dieses Verfahren eingegangen werden, der Interessierte sei auf [Böh05] und [DDL<sup>+</sup>90] verwiesen, wo es ausführlich erklärt wird. Man versucht bei der Verkleinerung möglichst wenig Informationen zu verlieren, und fasst deshalb einerseits Dokumente mit ähnlichem Inhalt, und andererseits Synonyme oder Terme, die häufig im selben Kontext erscheinen, zusammen. So werden quasi nebenbei Dokumentencluster erzeugt und Synonyme erkannt.

**Ablauf einer Anfrage** Um eine Anfrage mit LSI auszuwerten, wird diese selbst als ein Dokument betrachtet und ähnlich wie die Term-Dokument-Matrix in eine reduzierte Form transformiert. Zur Matrix in Abbildung 9 ist eine beispielhafte Anfrage in Abbildung 10 dargestellt. Zur Auswertung einer Anfrage wird zusätzlich noch eine weitere Matrix benötigt, welche die *Term-Konzept-Relation* widerspiegelt. Diese Matrix entsteht im Zuge der Singulärwertzerlegung und Reduktion. Ihre Zeilen stehen für Terme, ihre Spalten für Konzepte. Ein Beispiel für eine solche Matrix befindet sich in Abbildung 11, wobei die ersten drei Terme dem linken, die letzten zwei dem rechten Konzept entsprechen. Um nun zu erfahren, welchem Konzept die Anfrage ähnelt, wird ihr Dokumentenvektor mit der Matrix wie in Abbildung 12 multipliziert. Das Ergebnis dort sagt aus,

	D1	D2	D3	D4	D5	D6	D7
Maserati	1	2	1	5	0	0	0
Maybach	1	2	1	5	0	0	0
Mazda	1	2	1	5	0	0	0
Gen	0	0	0	0	2	3	1
Protein	0	0	0	0	2	3	1

Abbildung 9. Term-Dokument-Matrix

dass die Anfrage dem linken Konzept entspricht. Im weiteren Verlauf der Anfrageauswertung werden nun die Dokumente, welche ebenfalls das entsprechende Konzept behandeln, gemäß einer Abstandsbestimmung zur Anfrage, wie man sie auch aus dem Vektorraummodell kennt, gerankt. In realen Anwendungen des LSI werden natürlich nicht so schöne Ergebnisse wie in dem hier vorgestellten, konstruierten Beispiel herauskommen. Insbesondere gibt es mehr Konzepte und die Zugehörigkeit einer Anfrage zu diesen ist nicht so deutlich erkennbar.

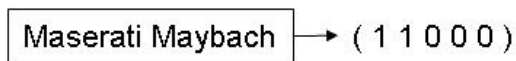


Abbildung 10. Beispiel einer Anfrage mit zugehörigem Vektor

$$\begin{pmatrix} 0.58 & 0 \\ 0.58 & 0 \\ 0.58 & 0 \\ 0 & 0.71 \\ 0 & 0.71 \end{pmatrix}$$

Abbildung 11. Term-Konzept-Relation

Die Dauer einer Anfrageauswertung ist aufgrund der nötigen Vektor-Matrix-Multiplikationen relativ hoch. Darüberhinaus hängt sie stark von der Anzahl der Dimensionen, auf die die Term-Dokument-Matrix reduziert wird, ab.

$$(1\ 1\ 0\ 0\ 0) \cdot \begin{pmatrix} 0.58 & 0 \\ 0.58 & 0 \\ 0.58 & 0 \\ 0 & 0.71 \\ 0 & 0.71 \end{pmatrix} = (1.16\ 0)$$

**Abbildung 12.** Vektor-Matrix-Multiplikation während der Anfrageauswertung

Es kann sich trotzdem lohnen, dieses relativ langsame Verfahren zu verwenden, da es semantische Zusammenhänge erkennt. Dadurch ist die Qualität der Ergebnisse besser als bei anderen Modellen, und es ist sogar möglich, relevante Dokumente zu finden, die keinen einzigen der Anfrageterme beinhalten.

**Größe** Die Größe einer mittels LSI indizierten Dokumentensammlung hängt einerseits von der Anzahl der Indexterme und der Anzahl der Dokumente ab, andererseits hat jedoch auch die Wahl der Dimensionsanzahl, auf die reduziert wird, einen großen Einfluss darauf.

**Retrieval-Modelle** Latent Semantic Indexing ist nicht für herkömmliche Retrieval-Modelle geeignet, da es einen eigenen Ansatz verwendet, welcher auf semantischen Beziehungen zwischen den Dokumenten einer Kollektion beruht.

**Kompression** Eine durch LSI indizierte Dokumentensammlung lässt sich nicht übermäßig gut komprimieren, da sie aus Matrizen und Vektoren besteht, die durch die Dimensionsreduktion schon in einer gewissen Weise komprimiert wurden. Außerdem kann sich eine zu starke Komprimierung schnell negativ auf die Dauer der Anfrageauswertung auswirken.

### 3 Schluss

#### 3.1 Zusammenfassung

In Abbildung 13 sind nochmal abschließend die wichtigsten Merkmale der vorgestellten Indizierungstechniken dargestellt. Man kann sehen, dass sich Invertierte Listen durch ihre Eignung für viele verschiedene Modelle, ihre in der Praxis sublineare Zeit- und Platzkomplexität sowie die sehr gute Komprimierbarkeit auszeichnen. Signatur-Dateien hingegen eignen sich hauptsächlich für kleinere Dokumentenkollektionen, da sie dann die schnellen Bitoperationen ausnutzen können,

ohne dass die lineare Zeitkomplexität sie ausbremst. Die Suffix-Arrays haben ihre Stärke in der Möglichkeit, auch nicht-textuelle Datenbestände gut indizieren zu können, was beispielsweise in der Biologie Verwendung findet. Die scheinbar schnelle Anfrageauswertung wird leider durch viele wahlfreie Textzugriffe behindert. Das LSI schließlich bezeichnet sogar ein eigenes Retrieval-Modell, dessen Stärke vor allem im Erkennen semantischer Zusammenhänge liegt.

	Invertierte Listen	Signatur-Dateien	Suffix-Arrays	LSI
Boolesches R.	ja	ja	ja	nein
Algebraisches R.	ja	nein	nein	nein
Fuzzy R.	ja	nein	nein	nein
Anfragedauer	$O(n^{0.8})$	$O(n)$	$O(\log n)$	hoch
Größe	$O(n^{0.85})$	$O(n)$	$O(n)$	dimensionsabhängig
Kompression	sehr gut	gut	schlecht	normal

Abbildung 13. Überblick über die Indizierungstechniken

### 3.2 Ausblick

In letzter Zeit ist die Größe der Datenbestände, welche indiziert werden sollen, rapide gewachsen, und es sieht so aus, als ob dieser Trend auch in naher Zukunft anhält. Andererseits wächst die Leistungsfähigkeit der Rechner und insbesondere die Geschwindigkeit der Prozessoren ebenso stark an. Im Zuge dieser Entwicklung nehmen die benötigten Ein- und Ausgabeoperationen mittlerweile einen großen Teil des Zeitaufwandes einer Anfrageauswertung für sich ein. Deshalb ist es ein Bestreben der Forschung, die Dauer und Häufigkeit dieser Operationen, wie beispielsweise Plattenzugriffe, zu reduzieren. Dabei spielt die Kompressionsfähigkeit eines Index eine große Rolle, weshalb es in diesem Bereich sicher noch einige Verbesserungen geben wird.

## Literatur

- [BCC94] Eric W. Brown, James P. Callan, and W. Bruce Croft. Fast incremental indexing for full-text information retrieval. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB'94)*, pages 192–202, Santiago de Chile, Chile, 12–15 September 1994. Morgan Kaufmann.
- [Böh05] Klemens Böhm. Vorlesung Datenbankeinsatz, 2005.
- [BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
- [DDL<sup>+</sup>90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [FC84] Chris Faloutsos and Stavros Christodoulakis. Signature files: an access method for documents and its analytical performance evaluation. *ACM Trans. Inf. Syst.*, 2(4):267–288, 1984.
- [Fer03] Reginald Ferber. *Information Retrieval*. dpunkt.verlag, 2003.
- [Gon87] Gaston Gonnet. Examples of pat applied to the oxford english dictionary. Technical Report OED-87-02, University of Waterloo, 1987.
- [HBYFL92] Donna Harman, R. Baeza-Yates, Edward Fox, and W. Lee. Inverted files. *Information retrieval: data structures and algorithms*, pages 28–43, 1992.
- [MM90] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 319–327, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.