Konzepte und Realisierung einer kontextbasierten Intranet-Suchmaschine

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

vorgelegt von

Christoph M. Mangold

aus Konstanz

Hauptberichter:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Mitberichter:	Prof. Dr. Ing. Stefan Deßloch
Tag der mündlichen Prüfung:	21.09.2007

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart
2007

Vorwort

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Abteilung Anwendersoftware des Instituts für Parallele und Verteilte Systeme (IPVS) der Universität Stuttgart. Inspiriert und motiviert wurde die Arbeit vor allem durch die interdisziplinäre Kooperation im Sonderforschungsbereich (SFB) 467 zum Thema der wandlungsfähigen Montage für die variantenreiche Serienproduktion.

Mein Dank gilt in erster Linie Herrn Prof. Bernhard Mitschang. Seine Offenheit für neue Ideen, seine vielfältigen Anregungen sowie die Freiheit, die er mir bei deren Umsetzung und Ausarbeitung gab, machten die Arbeit in dieser Form erst möglich. Der Gedankenaustausch durch zahlreiche Diskussionen und seine Unterstützung während der gesamten Bearbeitungszeit trugen wesentlich zum Gelingen der Arbeit bei. Herrn Prof. Deßloch möchte ich danken für das Interesse am Thema meiner Arbeit, für deren kritische Durchsicht, für seine Bereitschaft zur Übernahme des Mitberichts sowie für seine wohlwollende Beurteilung.

Ich danke allen aktuellen und ehemaligen Kolleginnen und Kollegen in der Abteilung Anwendersoftware für die nicht nur auf inhaltliche Aspekte beschränkte Zusammenarbeit und Unterstützung. Insbesondere danke ich Holger Schwarz für die zahlreichen fachlichen Ratschläge und Diskussionen sowie für die Geduld beim Korrekturlesen der Arbeit. Weiterhin danken möchte ich Stefan Berger und Sebastian Meyer für die inhaltlich fruchtbare und dennoch stets kurzweilige Zusammenarbeit im Sonderforschungsbereich. Vielen Dank an Benjamin Höferlin, der im Rahmen einer Hiwi-Tätigkeit wertvolle Hilfe bei der GUI des Prototyps sowie der Aufbereitung der Testdaten leistete. Nicht vergessen werden dürfen an dieser Stelle all diejenigen, die die Konzentration auf inhaltliche Fragestellungen erst ermöglichen. Insbesondere geht mein Dank an Annemarie Roesler, Manfred Rasch und Ralf Aumüller.

Ohne die Unterstützung meiner Familie wäre die Arbeit sicher nicht zustande gekommen. Vielen Dank für Eure Unterstützung, Eure Geduld und Euer Verständnis.

Stuttgart, im Dezember 2006 Christoph Mangold

Zusammenfassung

Suchmaschinen sind ein wichtiges Werkzeug zur Recherche von Dokumenten – nicht nur im World Wide Web, sondern gleichermaßen im Intranet von Unternehmen. Herkömmliche Dokumentensuchmaschinen werten zur Beantwortung von Suchanfragen lediglich den Inhalt, d.h. den Text der Dokumente aus. Der Ansatz der vorliegenden Arbeit basiert darauf, dass nicht nur der Text sondern ebenfalls der Kontext der Dokumente in die Auswertung miteinbezogen wird. Die Kontextinformation der Dokumente wird dazu aus den Datenbanken des Unternehmens extrahiert. Die kontextbasierte Suche ist dabei nicht als Alternative zu herkömmlicher, textbasierter Suche zu sehen, sondern als eine Erweiterung. Wie bei vielen Suchmaschinen üblich, spezifiziert der Benutzer den jeweiligen Informationsbedarf nicht als Ausdruck einer formalen Sprache, sondern als Schlüsselwortanfrage.

Zur Bestimmung der Dokumentenkontexte und als Abstraktion von Unternehmensdatenbanken wird ein graphenbasiertes Modell eingeführt, der ContextGraph. Die Knoten des ContextGraph repräsentieren einerseits Datenbankdaten und andererseits die vom System erfassten Dokumente. Die Kanten des ContextGraph modellieren Fremdschlüsselbeziehungen bzw. Beziehungen zwischen Tupeln und Attributwerten in der Datenbank. Jede Kante ist gewichtet mit einem Maß für den inhaltlichen bzw. semantischen Abstand der beiden Knoten die durch sie verbunden sind. Der ContextGraph bildet die Basis zur Berechnung des Kontexts von Dokumenten, welcher durch eine inkrementelle Kürzeste-Wege-Suche im ContextGraph bestimmt wird.

Bei der Bearbeitung von Suchanfragen und bei der Bewertung der Resultate wird nicht nur der Text sondern zusätzlich der Kontext von Dokumenten, d.h. die im Kontext der Dokumente enthaltenenen Begriffe berücksichtigt. Um dies zu ermöglichen werden Bewertungsmaße für die kontextbasierte Relevanz von Dokumenten bzgl. Suchbegriffen, für die kontextbasierte Wichtigkeit von Dokumenten und für die kontextbasierte Ähnlichkeit von Dokumenten entworfen. Diese Bewertungsmaße werden umgesetzt als eine Erweiterung des im Suchmaschinenbereich bewährten tf.idf-Bewertungsmaßes zur Bestimmung der Begriffsgewichte im Vektorraummodell.

Um den Ansatz praktisch zu erproben wird eine Architektur entworfen und darauf aufbauend ein prototypisches System zur kontextbasierten Suche implementiert. Damit Skalierbarkeit erreicht werden kann, verfolgt die Suchmaschine den indexbasierten Ansatz. Zur Indexierungszeit wird der Datenbestand erhoben und in Daten-

strukturen, sog. Indexen, abgelegt, die eine effiziente Verarbeitung von Suchanfragen zur Anfragezeit unterstützen.

Das implementierte System wird anhand zweier Szenarien analysiert. Dafür werden jeweils alternative Implementierungen der kontextbasierten Suche mit einer Implementierung der rein textbasierten Suche verglichen. Besonderes Augenmerk gilt dabei der Skalierbarkeit des Systems und einem Parameter zur Einstellung der vom System beachteten Kontextgröße. Die Messergebnisse quantifizieren einerseits den durch die Betrachtung des Kontexts nötigen Mehraufwand gegenüber der Textsuche. Andererseits wird die Qualität der Suchergebnisse analysiert. Die Auswertung der Messergebnisse belegen einen moderaten durch die Beachtung des Kontexts hervorgerufenen Mehraufwand, der sich – je nach Implementierung der Indexstrukturen – mehr im Aufwand zur Bearbeitung von Suchanfragen oder mehr im Aufwand bei der Erstellung des Index niederschlägt. In beiden analysierten Szenarien ergibt sich demgegenüber jedoch eine durch die Beachtung von Kontextinformation deutliche Verbesserung der Qualität der Suchresultate.

Abstract

Search engines are important tools for document retrieval. This is not only true in the World Wide Web, but also in the enterprise intranet. To process search requests and, in particular, rank the search results, standard search engines use mainly the text, i.e. the content of documents. In this work, we present an approach that is not restricted to document content, but additionally exploits document context. To this end, we derive the context of documents from the enterprise's databases. We do not consider the context-based document retrieval to be an alternative to common, text-based retrieval but rather to be an extension. As in many popular and established search engines the user specifies his information need not as a formal-language expression but as a list of search keywords.

As an abstraction of enterprise databases and as a basis to compute document contexts, we introduce a graph-based data model, which we call ContextGraph. On the one hand, ContextGraph nodes represent database entities and attribute values. On the other hand, they also model documents. Edges in the ContextGraph represent foreign-key relationships and relationships between tuples and attribute values in the database. Edges are directed and carry a measure that denotes the directed semantic distance between their two adjacent nodes. Based on the ContextGraph we determine the contexts of documents by means of an greedy shortest-paths algorithm.

To process search requests and to rank the search results we do not only consider keywords from the documents' content but also from their context. To this end we introduce ranking measures for context-based relevance of documents concerning a given keyword search query, for context-based document importance, and for context-based document similarity. These measures we implement as an extension to the tf.idf measure that calculates term-weights in the vector-space model. Both, tf.idf, and the vector-space model are well-known and proven concepts in the area of information retrieval.

As a proof of concept, we present an architecture and explain how we implemented our prototype system that realizes a context-based search engine. To enable scalability the system follows the index-based approach. At indexing time, the it analyzes the enterprise's data and builds a data structure, the index, that supports efficient query processing at query time.

We analyze and evaluate our system based on two scenarios. To this end, we compare three alternative implementations of the context-based search engine with a

pure text-based search engine. In particular, we focus on the scalability of the system and on the influence of an important parameter that adjusts the size of the evaluated context. On the one hand, our measurements quantify the overhead of context-based search as opposed to pure text-based search. On the other hand, we analyze the quality of search results. Evaluations of our measurements show that the performance overhead is moderate. Depending on the implementation of the index, the emphasis of overhead can be placed either at indexing time or at query time. Furthermore, our evaluations on both analyzed scenarios demonstrate that the overhead is not in vain. Compared with the text-based search engine, the context-based engine shows a significant increase in the quality of search results.

Inhaltsverzeichnis

Inhaltsverzeichnis **9**

1 Einleitung **13**
1.1 Beispiel 15
1.2 Anforderungen an eine Intranetsuchmaschine 17
1.3 Ansatz und System im Überblick 18
1.4 Ausgeklammerte Problembereiche 19
1.5 Überblick über die Arbeit 20

2 Grundlagen **25**
2.1 Dokumentensuche 25
2.1.1 Aufbau 26
2.1.2 Indexe 27
2.1.3 Bewertung von Suchresultaten 28
2.1.4 Bewertung der Qualität einer Suchmaschine 33
2.1.5 Bedeutung für den Entwurf 33
2.2 Schlüsselwortsuche in relationalen Datenbanken 34
2.2.1 DataSpot 35
2.2.2 Jungle 35
2.2.3 BANKS 36
2.2.4 DBXplorer und DISCOVER 37
2.2.5 Am Markt verfügbare Produkte 38
2.2.6 Bedeutung für den Entwurf 39
2.3 Suche in semistrukturierten Daten 40
2.3.1 Schlüsselwortsuche auf einer Lorel-Datenbank 41
2.3.2 SphereSearch 41
2.3.3 Bedeutung für den Entwurf 42
2.4 Integration strukturierter und unstrukturierter Daten für die Suche . 43
2.4.1 Kopplung von Datenbank und Suchmaschine 43
2.4.2 WSQ 43
2.4.3 SCORE 44
2.4.4 RDBMS als Suchmaschine 44
2.4.5 BIKM 45

2.4.6 Bedeutung für den Entwurf 45
2.5 Semantische Suche 46
2.5.1 Semantic Web und Ontologien 46
2.5.2 Merkmale semantischer Suche 48
2.5.3 Abgrenzung und ausgewählte Ansätze 50
2.5.4 Kategorisierung und Fazit 55
2.5.5 Bedeutung für den Entwurf 56
2.6 Wissen im Unternehmen 57
2.6.1 Das Stuttgarter Wissensmodell 58
2.6.2 Bedeutung für den Entwurf 59
2.7 Graphenbasierte Abstandsmaße 60
2.7.1 Spreading Activation 60
2.7.2 Kürzeste Wege 62
2.7.3 Bedeutung für den Entwurf 64
2.8 Zusammenfassung 65

3 Konzepte kontextbasierter Suche 69
3.1 Der ContextGraph 69
3.1.1 Definition 69
3.1.2 Einbettung relationaler Datenbanken 72
3.1.3 Semantischer Abstand und Kontext 76
3.2 Bewertungsmaße 86
3.2.1 Relevanz eines Dokuments 87
3.2.2 Wichtigkeit eines Dokuments 101
3.2.3 Suche nach ähnlichen Dokumenten 103
3.3 Zusammenfassung 107

4 Architektur und Prototyp 109
4.1 Architektur und Überblick 109
4.2 Import und ContextGraph 113
4.2.1 Eingesetzte Bibliotheken 113
4.2.2 Import-Komponenten 114
4.2.3 Die ContextGraph-Komponente 115
4.3 Kantengewichtung 115
4.4 ObjectRank, Kürzeste Wege und Dokumentenimport 116
4.5 Kontextualisierung 118
4.6 Indexerzeugung und Indexauswertung 118
4.6.1 Lucene 119
4.6.2 Einzelschrittindex 123
4.6.3 Doppelschrittindex für Schlüsselwortsuche 128
4.6.4 Doppelschrittindex für Ähnlichkeitssuche 132
4.7 Semantische Suche 135

4.8 Benutzerschnittstelle 136
4.9 Zusammenfassung 137

5 Analyse und Bewertung 141
5.1 Szenarien und Messumgebung 141
5.1.1 Maschinenbauszenario 142
5.1.2 Citeseerszenario 143
5.1.3 Messumgebung 144
5.2 Qualität der Suchresultate 145
5.2.1 Das Qualitätsmaß Precision@10 145
5.2.2 Schlüsselwortsuche 147
5.2.3 Suche nach ähnlichen Dokumenten 151
5.3 Indexgröße 153
5.4 Berechnung der kürzesten Wege 156
5.5 Erstellung des Index 157
5.6 Suchanfragen 160
5.6.1 Schlüsselwortsuche 160
5.6.2 Suche nach ähnlichen Dokumenten 161
5.7 Zusammenfassung 164

6 Zusammenfassung und Ausblick 167
6.1 Zusammenfassung 167
6.2 Einordnung des Systems 168
6.3 Abdeckung der Anforderungen 169
6.4 Ausblick 170

Abbildungsverzeichnis 173

Tabellenverzeichnis 177

Literaturverzeichnis 179

1

Einleitung

Die Datenmenge wächst. Dies trifft auf sämtliche informationstechnisch erfassten Bereiche zu. Schon längst sind Größenordnungen erreicht, die weit jenseits der vom menschlichen Benutzer erfassbaren Informationsmenge liegen. Mit dem Anwachsen der Datenmenge tritt zunehmend die Aufgabe der effizienten Recherche von Informationen bzw. Dokumenten (engl.: Information Retrieval) in den Vordergrund. So verschwenden Mitarbeiter europäischer Unternehmen durchschnittlich 67 Minuten am Tag mit der Suche nach Information.

Die vorliegende Arbeit befasst sich mit der Recherche von Dokumenten im Unternehmen, einem Bereich, in dem infolge zunehmender Automatisierung Datenmengen unvorstellbarer Größe erhoben und gespeichert werden. In zunehmendem Maße setzt sich hier jedoch die Erkenntnis durch, dass Erhebung und Speicherung von Daten alleine nicht ausreichend sind. Ohne effizienten und intuitiven Zugriff ist die abgelegte Information de facto nutzlos und teuer. Dass dieses Problem insbesondere im Intranet ernst genommen wird, zeigen die Lösungen großer Hersteller, wie z.B. der IBM Net Search Extender [IBM, 2006], IBM Omnifind [Buehler, 2005], Oracle Ultra Search [Oracle, 2006], SAP Netweaver [Computerwoche, 2006] oder Google Enterprise Search [Google, 2003].

Die im Unternehmen abgelegte Information zerfällt in zwei Teile. Auf der einen Seite enthalten *Dokumente* unstrukturierte oder semistrukturierte Textinformation. Dokumente beinhalten Berichte, Pläne, Protokolle, Webseiten, Emails und vieles mehr. Sie werden üblicherweise gemäß unternehmensspezifischer Arbeitsabläufe (engl.: Workflows) erstellt, bearbeitet und abgelegt. Auf der anderen Seite steht die in *Datenbanken* strukturiert abgelegte Information, die über Unternehmensinformationssysteme, wie z.B. Produktionsplanungs- oder Warenwirtschaftssysteme, verfügbar gemacht werden.

Inhaltlich besteht ein sehr enger Zusammenhang zwischen den Informationen in Dokumenten und in Datenbanken. In der Regel ergänzen sie sich inhaltlich und

bilden gemeinsam die Informationsbasis des Unternehmens. Beide Arten von Information sind im Unternehmen somit unabdingbar.

Auf der Ebene der Informations*systeme* zeigt sich, dass dieser starke inhaltliche Zusammenhang nur unzureichend reflektiert wird. Das Wiederauffinden von Information auf der Seite der Dokumente wird durch Volltextsuchmaschinen realisiert, über die allenfalls per Suchformular bestimmte, fest definierte Dokumentenmetadaten, wie z.B. Autor oder Dokumententitel, in die Suche miteinbezogen werden können. Auf der Datenbankseite wird der vom Benutzer meist durch Suchformulare spezifizierte oder vom Anwendungsprogrammierer festgelegte Informationsbedarf letztendlich durch Anfragesprachen, wie z.B. SQL, an die Datenbank übermittelt. Eine Verknüpfung der Information beider Seiten findet in der Regel nicht statt.

Aus unternehmerischer Sicht bedeutet die auf Systemebene fehlende Verknüpfung der im Unternehmen verfügbaren Information, dass die intellektuelle Leistung, welche zur Verknüpfung komplementärer Information aufzuwenden ist, von jedem Mitarbeiter separat geleistet werden muss. Die eigentliche Verknüpfung der Informationen hat somit in den Köpfen der Mitarbeiter stattzufinden. Dieser Prozess ist oftmals gleichermaßen langwierig wie kostenintensiv. Um den Prozess zu beschleunigen und somit Kosten einzusparen, werden in vielen Unternehmen Systeme zur Unterstützung des betrieblichen Wissensmanagements eingesetzt, vgl. [Huber, 2005; Sandkuhl, 2005; Gottschick et al., 2005].

Die Grundidee der vorliegenden Arbeit wurde im Rahmen des Sonderforschungsbereichs 467 zum Thema wandlungsfähiger Unternehmensstrukturen erarbeitet. Eine in diesem Rahmen durchgeführte empirische Untersuchung ergab, dass die effektive Erzeugung von Verknüpfungen zwischen inhaltlich zusammenhängender Information – dem sogenannten Integrationswissen – insbesondere von Unternehmen im turbulenten Umfeld als wesentlicher Erfolgsfaktor eingestuft wird [Meyer, 2007]. Auf die Bedeutung von Wissen im Unternehmen wird in Abschnitt 2.6 weiter eingegangen.

Der Ansatz der vorliegenden Arbeit beruht darauf, vorhandene Verbindungen zwischen Dokumenten und Datenbanken im Unternehmen für die Dokumentenrecherche auszunutzen und so die Notwendigkeit zur Verknüpfung der Informationen in den Köpfen der Mitarbeiter zu minimieren. Dazu wird bei der Suche nach Dokumenten nicht mehr lediglich der *Inhalt* der Dokumente beachtet, sondern gleichermaßen ihr *Kontext*. Die Information über den Kontext eines Dokuments wird dazu aus der zum Dokument komplementären Information der Unternehmensdatenbanken abgeleitet.

Im Rahmen der Semantic-Web-Initiative entstanden verschiedene Ansätze zur sogenannten "semantischen" Suche. Die Kernidee bei der semantischen Suche besteht zumeist darin, den Suchprozess durch formalisiertes Kontextwissen zu unterstützen. Kontextwissen wird dazu in Wissensstrukturen – sogenannten Ontologien – abgelegt. Auf diese Weise kann beispielsweise das Problem der Mehrdeutigkeit von Begriffen automatisch behandelt werden. Die Suchmaschine ist dann bspw. in der

Lage bei der Suche nach “Lager” selbstständig zu entscheiden, ob Kugellager oder Materiallager gemeint ist. Die im Rahmen des Semantic Web entstandenen Ansätze setzen fast durchweg von Domänenexperten manuell erstellte und gewartete Wissensstrukturen voraus. Eine detaillierte Betrachtung verwandter Arbeiten auf diesem Gebiet findet sich in Abschnitt 2.5. Übertragen auf das Szenario der Suche im Unternehmen bedeutet dies, dass die inhaltlichen Verknüpfungen von einer Fachkraft aufwändig in einem System modelliert, eingepflegt und gewartet werden müssten. Die vorliegende Arbeit geht von der Annahme aus, dass viele dieser inhaltlichen Verknüpfungen bereits in den Datenbanken des Unternehmens abgelegt sind. Dadurch wird keine aufwändig zu unterhaltende Unternehmensontologie erforderlich, sondern anstelle dessen auf die stets aktuelle und konsistente Information der Unternehmensdatenbanken zurückgegriffen.

Der weitere Verlauf des Kapitels gliedert sich wie folgt: Um die der Arbeit zugrunde liegende Idee zu illustrieren, wird in Abschnitt 1.1 zunächst ein Beispielszenario einer Suchanfrage im Intranet gegeben. Abschnitt 1.2 formuliert die allgemeinen Anforderungen an eine Suchmaschine im Intranet. Darauf aufbauend wird in Abschnitt 1.3 ein inhaltlicher Überblick über die im Rahmen der vorliegenden Arbeit erarbeiteten Konzepte sowie das prototypisch implementierte System gegeben. Um Missverständnisse früh auszuschließen, werden in Abschnitt 1.4 Problembereiche benannt, die zwar inhaltlich mit der vorliegenden Arbeit verwandt sind, jedoch im weiteren nicht diskutiert werden. Abschnitt 1.5 gibt schließlich einen kapitelweisen Überblick über den restlichen Aufbau der Arbeit.

1.1 Beispiel

Um die der vorliegenden Arbeit zugrunde liegende Idee zu illustrieren, sei folgendes Beispiel gegeben, auf welches in der weiteren Ausarbeitung mehrfach Bezug genommen werden wird.

In einem produzierenden Unternehmen werden Fehlerberichte in einem Dateisystem abgelegt und im Dokumentenmanagementsystem (DMS) katalogisiert. Im datenbankbasierten Produktionsplanungssystem (PPS) wird gespeichert, welche Produkte auf welchen Maschinen gefertigt werden.

Ein kleines Beispielszenario ist in Abbildung 1.1 dargestellt. Das Szenario enthält einen einzigen Fehlerbericht, der einen Klebefehler bei der Produktion von Hubhebeln auf der Maschine M18 beschreibt. Der Bericht ist im Dateisystem abgelegt. Im DMS werden zusätzlich Verweise auf die betroffene Maschine (M18) und auf das betroffene Produkt (Hubhebel) sowie der Typ des Dokuments (Fehlerbericht) und eine Kurzbeschreibung des Berichts gespeichert. Das PPS gibt Aufschluss darüber, dass sowohl das Produkt Hubhebel als auch das Produkt Schwinghebel auf Maschine M18 gefertigt werden.

Man betrachte die folgende Situation: Ein Mitarbeiter der Qualitätssicherungs-

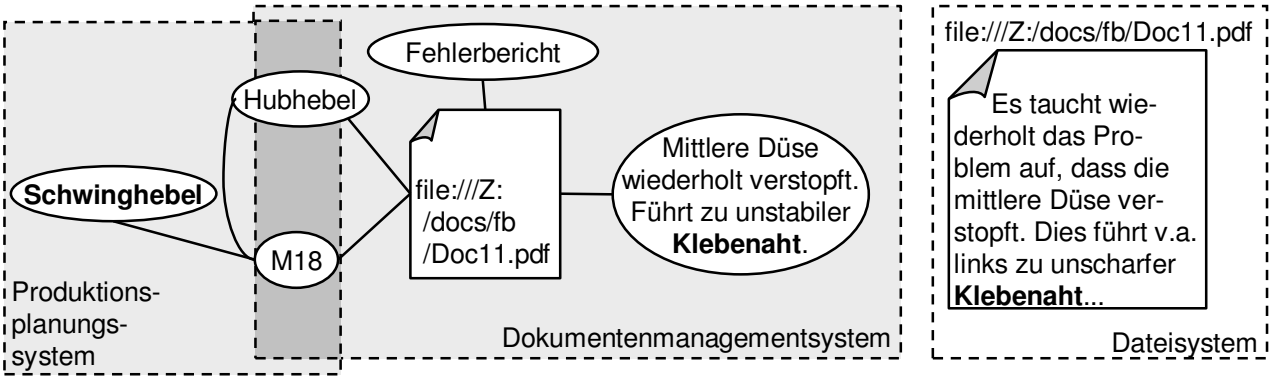


Abbildung 1.1: Beispielszenario.

abteilung entdeckt eine fehlerhafte Klebenaht beim Produkt Schwinghebel. Um herauszufinden, ob dieser oder ähnliche Fehler zuvor auftraten, gibt er die Suchanfrage "Schwinghebel UND Klebenaht" in seine Dokumentensuchmaschine ein, in der alle im DMS verzeichneten Dokumente erfasst sind. Falls kein Dokument existiert, welches beide gesuchten Begriffe, Schwinghebel und Klebenaht, enthält, wird eine herkömmliche, textbasierte Dokumentensuchmaschine viele Dokumente finden, die *entweder* Schwinghebel *oder* Klebenaht enthalten. Mit dieser potenziell sehr großen Menge an Antwortdokumenten konfrontiert, bieten sich dem Mitarbeiter mehrere Möglichkeiten für das weitere Vorgehen.

Ein potenziell aufwändiger Ansatz zur Recherche relevanter Dokumente besteht in der manuellen Durchsicht und Bewertung aller Suchresultate. Sehr viel wahrscheinlicher wird der Mitarbeiter zuvor versuchen, mit einer Umformulierung der Anfrage zum Ziel zu gelangen. Für diese Umformulierung können, so vorhanden, vom Mitarbeiter gelernte, inhaltliche Verknüpfungsinformationen herangezogen werden. Sind diese in nicht ausreichendem Umfang vorhanden, bleibt der Abruf dieser Informationen aus Unternehmensdatenbanken oder die Nachfrage bei erfahreneren Kollegen. Der in dieser Arbeit vorgestellte Ansatz zielt darauf ab, die hierfür notwendige Kontextinformation direkt aus den Unternehmensdatenbanken abzugreifen und zur Verarbeitung der Suchanfrage heranzuziehen.

Für obiges Beispiel bedeutet dies, dass das Ziel verfolgt werden muss, das dargestellte Dokument weit oben in der Liste der Resultate anzuzeigen. Es behandelt zwar keinen Fehler der Klebenaht von Schwinghebeln, dafür jedoch ein ähnliches Problem bei Hubhebeln, welche auf derselben Maschine gefertigt werden wie Schwinghebel. Es besteht daher die Möglichkeit, dass die bei der Fertigung von Hubhebeln betrachteten Probleme auf Schwinghebel übertragbar sind.

In den Informationssystemen des Unternehmens schlägt sich dieser inhaltliche Zusammenhang darin nieder, dass das Dokument zwar – genau wie alle anderen gefundenen Dokumente – nur einen der beiden gesuchten Begriffe enthält. Der andere Begriff kommt jedoch im Kontext des Dokuments vor.

Die informationstechnische Verknüpfung von Dokument und Kontextinformation kann also bei der Dokumentensuche nützlich sein. Der inhaltliche Zusammenhang

von Schwinghebeln und Hubhebeln, der daraus resultiert, dass beide auf derselben Maschine gefertigt werden, muss nicht länger vom Mitarbeiter gelernt werden. Vielmehr wird dieser Zusammenhang automatisch aus den Datenbanken des Unternehmens abgeleitet und kann so zur Verbesserung des Suchprozesses beitragen.

Nicht zufällig wurde in Abbildung 1.1 eine Graphendarstellung zur Illustration der inhaltlichen Zusammenhänge gewählt. Für den Rest dieser Arbeit wird ein graphenbasiertes Datenmodell als Abstraktion von Unternehmensinformationssystemen verwendet. Da dieser Graph den Kontext der Dokumente eines Unternehmens modelliert, wird er im Folgenden *ContextGraph* genannt.

1.2 Anforderungen an eine Intranetsuchmaschine

Aus dem im letzten Abschnitt beschriebenen Szenario lassen sich folgende allgemeine Anforderungen an eine Intranetsuchmaschine ableiten, siehe dazu auch [Mangold et al., 2005].

1. Schlüsselwortsuche und Suche nach ähnlichen Dokumenten. Es ist eine möglichst einfache Benutzerschnittstelle bereitzustellen. Nach Jansen et al. [1998], ist der durchschnittliche Websuchmaschinenbenutzer nur selten willens, erweiterte Suchmöglichkeiten, wie z.B. die boolesche Verknüpfung von Suchbegriffen, zu verwenden. Daher muss die Formulierung der Suchanfrage durch Schlüsselworte möglich sein. Die Formulierung eines formalen Ausdrucks, der den Anforderungen einer strukturierten Anfragesprache gehorcht, entspricht diesen Anforderungen nicht. Zusätzlich zur Schlüsselwortsuche soll die Suche nach ähnlichen Dokumenten unterstützt werden.

2. Transparenz. Die Studie von Jansen et al. [1998] beschreibt am Beispiel von Relevance Feedback, dass Benutzer nur relativ selten willens sind, zur Verbesserung ihrer Suchergebnisse zusätzliche Angaben zu machen oder sogar einen iterativen Prozess in Kauf zu nehmen. Die Interna einer Suchmaschine sollten dem Benutzer möglichst verborgen bleiben. Bereits bei der ersten Anfrage sollte die Maschine gute Resultate liefern und nicht auf Rückfragen beim Benutzer angewiesen sein.

3. Anpassbarkeit und Erweiterbarkeit. Eine Suchmaschine kann kaum für alle Szenarien in allen Unternehmen gleichermaßen geeignet sein. Es ist daher erforderlich, ein offenes System zu entwerfen, welches sich flexibel an verschiedene Situationen anpassen und leicht erweitern lässt.

4. Textsuche. Wie sich am Beispiel in Abschnitt 1.1 leicht ablesen lässt, ist die Auswertung von Dokumentenkontext nicht als Alternative zur Volltextsuche zu sehen, sondern vielmehr als eine Erweiterung. Daher ist die im Bereich des

Information Retrieval entwickelte Technologie keinesfalls abzulösen, sondern sinnvoll um kontextbasierte Aspekte anzureichern.

5. Performanz und Skalierbarkeit. Eine Suchmaschine sollte vom Benutzer interaktiv verwendet werden können. Suchanfragen sollten daher auch für große Datenmengen in weniger als einer Sekunde beantwortet werden können.

6. Qualität. Neben der Performanz einer Suchmaschine ist gleichermaßen die Qualität ihrer Suchergebnisse ausschlaggebend. Diese Qualität misst sich i. Allg. an der Relevanz der gefundenen Dokumente bzgl. des Informationsbedarfs des Benutzers. Üblicherweise werden von Dokumentensuchmaschinen sehr viel mehr Dokumente für eine Suchanfrage gefunden, als ein Benutzer jemals lesen könnte oder wollte. Oft werden vom Benutzer daher nur solche Dokumente beachtet, die sich unter den ersten zehn oder 20 Resultaten befinden [Jansen et al., 1998; Wolfram, 2000]. Es ist daher von größter Wichtigkeit, dass die für den Benutzer relevantesten Dokumente zu Beginn der Resultatsliste angezeigt werden. Die Sortierung der Suchresultate wird im Information Retrieval durch Bewertungsmaße (engl.: Ranking) realisiert. Die Qualität einer Suchmaschine bzw. ihrer Suchresultate hängt somit im Wesentlichen von der eingesetzten Bewertungsstrategie ab.

7. Integration. Wie bereits am Beispiel diskutiert, besteht der Mehrwert einer Intranetsuchmaschine, wie in dieser Arbeit beschrieben, darin, strukturierte Daten für die Recherche unstrukturierter Daten zu erschließen. Wie das Beispiel oben illustriert, stehen dabei vor allem die im Unternehmen bereits vorhandenen Datenbanken im Vordergrund. Analog zu strukturierten Datenbanken sollen ebenfalls semistrukturierte Datenquellen sowie im Unternehmen vorhandene Wissensstrukturen integriert werden können.

1.3 Ansatz und System im Überblick

Bevor in Kapitel 2 einige Grundlagen sowie die zur vorliegenden Arbeit verwandten Ansätze im Detail diskutiert werden, wird an dieser Stelle zum besseren Verständnis der Ansatz und das System der vorliegenden Arbeit im Überblick vorgestellt. Dabei wird teilweise direkt auf die in Abschnitt 1.2 formulierten Anforderungen Bezug genommen.

Der Ansatz zur kontextbasierten Intranet-Suchmaschine ist, wie in Anforderung (4) formuliert, als eine Erweiterung herkömmlicher Textsuche zu verstehen. Nach Anforderung (1) handelt es sich dabei um Schlüsselwortsuche.

Der Kontext eines Dokuments wird nach Anforderung (7) aus den Datenbanken des Unternehmens generiert. Dazu wird, wie in Abschnitt 1.1 angedeutet, ein graphenbasiertes Datenmodell, der ContextGraph, eingeführt, welches als Abstraktion

der zugrundeliegenden Datenbanken dient. Die Knoten des ContextGraphs repräsentieren dabei sowohl strukturierte Daten, als auch Dokumente. Die Kanten des Graphen werden gewichtet mit einem Maß für den inhaltlichen, bzw. semantischen Abstand der beiden Knoten, die durch die Kante verbunden sind. Der so gewichtete ContextGraph bildet die Grundlage zur Bestimmung der Dokumentenkontexte.

Die vom ContextGraph abgeleitete Kontextinformation wird bei der Berechnung der Bewertungsmaße der Suchresultate berücksichtigt. Dazu werden im Information Retrieval gängige Bewertungsmaße aufgegriffen und auf den kontextbasierten Fall erweitert.

Um die Flexibilität und Erweiterbarkeit der Suchmaschine nach Anforderung (3) zu gewährleisten, wurde eine komponentenbasierte Architektur gewählt. Damit Skalierbarkeit nach Anforderung (5) erreicht werden kann, verfolgt die Suchmaschine den indexbasierten Ansatz. Zur Indexierungszeit wird der Datenbestand erhoben und in Datenstrukturen, sog. Indexen, abgelegt, die eine effiziente Verarbeitung von Suchanfragen zur Anfragezeit unterstützen. Bei der Implementierung wurde nach Anforderung (4) darauf geachtet, dass die Suchmaschine gleichzeitig Dokumente mit und ohne Kontext verarbeiten kann und somit eine echte Erweiterung herkömmlicher Textsuche darstellt.

Bei der Analyse des implementierten Systems wurde einerseits der durch die Betrachtung des Kontexts nötige Mehraufwand gegenüber herkömmlicher Textsuche quantifiziert. Andererseits konnte gezeigt werden, dass sich dadurch in den analysierten Szenarien eine Verbesserung der Qualität der Suchresultate ergibt.

1.4 Ausgeklammerte Problembereiche

In der vorliegenden Arbeit werden eine Vielzahl unterschiedlicher Fachgebiete angesprochen. Es liegt auf der Hand, dass nicht alle Gebiete im Detail diskutiert werden können. Die im folgenden aufgezählten Bereiche grenzen an die in der vorliegenden Arbeit besprochenen Themengebiete an, werden jedoch nicht weiter besprochen.

- Integration von Dokumenten und Datenbanken. Die vorliegende Arbeit setzt ein Szenario voraus, in dem Dokumente durch ihre Metadaten bereits mit Information in Unternehmensdatenbanken verknüpft sind. Selbstverständlich trifft diese Annahme nicht für sämtliche Unternehmen zu. Ansätze, wie bestehende Dokumentenstämme mit strukturierten Daten integriert werden können, werden in der vorliegenden Arbeit nicht behandelt; es sei jedoch auf die BIKM-Technologie [Cody et al., 2002] und auf das EROCS-System [Chakaravarthy et al., 2006] verwiesen, die sich beide diesem Problembereich widmen.

- Textanalyse. Es existieren zahlreiche Arbeiten in den Bereichen der Textanalyse / Natürlichsprachanalyse und der Informationsextraktion. Einen guten Überblick zu diesem Themenkomplex gibt Manning und Schütze [1999]. Die vor-

liegende Arbeit befasst sich nicht mit Textverständnis, sondern mit der Kontextualisierung von Dokumenten. Es ist durchaus denkbar, das Ergebnis von Textanalyse-Algorithmen bei der Intranetsuche zu berücksichtigen. Die Analyse als solche liegt jedoch nicht im Fokus der vorliegenden Arbeit und wird aus diesem Grund nicht weiter diskutiert.

- Datenbankintegration. In der vorliegenden Arbeit wird weiterhin nicht auf Aspekte zur Integration von Datenbanken eingegangen. Der vorgestellte Ansatz fokussiert nicht darauf, Informationen aus verschiedenen Quellen zu integrieren. Hier wird auf die breite Palette an Ansätzen im Bereich der Datenbankintegration verwiesen. Da das vorgestellte System nur lesenden Zugriff auf die Daten erforderlich macht, können sämtliche Arten von Integrationslösungen von Data Warehouses [Inmon, 1996; Chaudhuri und Dayal, 1997; Bauer und Günzel, 2004] bis zu föderierten Datenbanken [Sheth und Larson, 1990; Conrad, 1997; Busse et al., 1999] als Datenquelle verwendet werden.

- Information Retrieval. Der Bereich Information Retrieval umfasst ein breites Spektrum an Algorithmen, Technologien und Produkten. Es liegt auf der Hand, dass im Rahmen der vorliegenden Arbeit kein umfassendes Bild dieses Bereichs gegeben werden kann. Nur solche Details werden diskutiert, die zum Verständnis der vorgestellten Ansätze erforderlich sind.

- Künstliche Intelligenz. Der in der vorliegenden Arbeit beschriebene Ansatz weist Parallelen auf zu Ansätzen im Bereich der semantischen Suche. Viele dieser Ansätze entstammen der künstlichen Intelligenz (KI). Zwar werden in Abschnitt 2.5 verschiedene dieser Ansätze vorgestellt, eine Diskussion der entsprechenden Grundlagen muss jedoch entfallen.

- Wissensmanagement. Die Suchmaschine kann als eine Maßnahme zur Unterstützung des betrieblichen Wissensmanagements verstanden werden. Auf die diesbezügliche Bedeutung wird kurz in Abschnitt 2.6 eingegangen. Weitere Aspekte zum Wissensmanagement, wie beispielsweise alternative betriebliche Maßnahmen zur Vernetzung von Informationsquellen, werden nicht diskutiert.

1.5 Überblick über die Arbeit

Dieser Abschnitt gibt einen kurzen Überblick über die verbleibenden Kapitel der vorliegenden Arbeit.

Kapitel 2: Grundlagen

Das zweite Kapitel befasst sich mit Grundlagen und verwandten Arbeiten. Um den Einfluss der vorgestellten Ideen und Technologien zu verdeutlichen, wird zum Ab-

schluss jedes Abschnitts die jeweilige Bedeutung für den Entwurf der vorliegenden Arbeit diskutiert.

Zunächst werden in Abschnitt 2.1 gängige Ansätze der Dokumentensuche betrachtet, insbesondere mit Hinblick auf den Aufbau von Suchmaschinen, auf die Bewertung von Suchresultaten, auf die Suche nach ähnlichen Dokumenten und auf die Bewertung der Qualität von Suchmaschinen. Die hier vorgestellten Ansätze und Technologien bilden aus Sicht des Information Retrieval das Grundgerüst der vorliegenden Arbeit.

Im Abschnitt 2.2 werden verwandte Arbeiten vorgestellt, die eine Volltextsuche in relationalen Datenbanken zum Ziel haben. Teilweise wird dazu die zu durchsuchende Datenbank ebenfalls als Graph repräsentiert.

Mit der zunehmenden Verbreitung des XML-Datenformats wuchs in den vergangenen Jahren das Interesse an effizienten Methoden zur Verarbeitung von XML-Daten. In Abschnitt 2.3 werden verwandte Arbeiten auf dem Gebiet der Suche in semistrukturierten Daten vorgestellt.

Die vorliegende Arbeit basiert auf der Idee, strukturierte Datenbankinformation mit unstrukturierten Dokumenten im Unternehmen zu verbinden. In Abschnitt 2.4 werden verwandte Arbeiten vorgestellt, die ebenfalls die Verknüpfung strukturierter und unstrukturierter Daten vornehmen.

Abschnitt 2.5 beschäftigt sich mit verwandten Arbeiten aus dem Bereich der semantischen Suche. Unter semantischer Suche wird i. Allg. die Unterstützung der Schlüsselwortsuche durch Wissensrepräsentationssysteme verstanden. Da auf diesem Gebiet eine Vielzahl von Ansätzen existiert, wird zunächst ein Kategorisierungsschema vorgeschlagen. Sodann werden ausgewählte Ansätze vorgestellt und anhand der Kategorisierung diskutiert und verglichen.

Die vorliegende Arbeit entstand im Rahmen eines interdisziplinären Projekts zum Wissensmanagement in wandlungsfähigen Unternehmen. Dementsprechend gibt Abschnitt 2.6 einen kurzen Überblick über das in diesem Rahmen entwickelte Wissensmodell und erklärt die Bedeutung des Ansatzes vor dem Hintergrund des betrieblichen Wissensmanagements.

In Abschnitt 2.7 werden Spreading Activation und kürzeste Wege als alternative graphenbasierte Abstandsmaße vorgestellt. Basierend auf der im ContextGraph modellierten Information können beide Ansätze verfolgt werden, um die gegenseitige Relevanz, bzw. den inhaltlichen Zusammenhang von Knoten im Graph zu bestimmen. Zur Berechnung der kürzesten Wege im ContextGraph wird kurz auf den Algorithmus von Dijkstra sowie auf den HiddenPath-Algorithmus eingegangen.

Zum Abschluss des Kapitels gibt Abschnitt 2.8 eine kurze Zusammenfassung der vorgestellten Arbeiten.

Kapitel 3: Konzepte kontextbasierter Suche

In Kapitel 3 wird das abstrakte Grundgerüst für die kontextbasierte Dokumentensuche entworfen. Der Fokus liegt dabei einerseits auf der Konstruktion des ContextGraph, andererseits darauf aufbauend auf der kontextbasierten Bewertung von Suchresultaten.

Im Abschnitt 3.1 wird zunächst eine Definition des ContextGraph gegeben sowie mögliche Abbildungen relationaler Datenbanken auf den ContextGraph diskutiert. Wie bereits angesprochen, sind die Kanten des Graphen mit einem Maß für semantischen Abstand annotiert. Der Abschnitt wird daher abgeschlossen mit einer Diskussion zur Bestimmung der Kantengewichtung sowie mit der darauf basierenden Definition und Berechnung des Kontexts von Dokumenten.

Die Bewertung von Suchresultaten ist ein wichtiges Qualitätsmerkmal von Suchmaschinen. Die Diskussion in Abschnitt 3.2 gliedert sich in Betrachtungen zur Wichtigkeit und zur Relevanz von Dokumenten bei der Schlüsselwortsuche und schließt mit einer Diskussion zur Bewertung von Resultaten bei der Suche nach ähnlichen Dokumenten. Beim Entwurf der kontextbasierten Bewertungsmaße stand im Vordergrund, die im Bereich des herkömmlichen Information Retrieval gewonnene Erfahrung aufzugreifen und auf ein kontextbasiertes Szenario anzuwenden. Bei der Diskussion zur Relevanz von Dokumenten werden daher zunächst abstrakt die Anforderungen an kontextbasierte Resultatsbewertung formuliert. Sodann werden bewährte Maße herkömmlicher Textsuche vorgestellt und eine Erweiterung auf den kontextbasierten Fall vorgeschlagen.

Kapitel 4: Architektur und Prototyp

Um die theoretisch erarbeiteten Konzepte praktisch zu erproben, wurde ein Prototyp implementiert, der in Kapitel 4 vorgestellt wird.

Die Erweiterbarkeit und Anpassbarkeit wird durch eine komponentenbasierte Architektur ermöglicht. Abschnitt 4.1 gibt einen Überblick über diese Architektur. Die darauf folgenden Abschnitte besprechen die einzelnen Komponenten im Detail, wobei einerseits deren Rolle in der Architektur und andererseits deren Implementierung diskutiert wird. Besonderes Augenmerk liegt dabei auf der Diskussion alternativer Indexstrukturen.

Kapitel 5: Analyse und Bewertung

In Kapitel 5 wird der im Rahmen der vorliegenden Arbeit implementierte Prototyp anhand von Messergebnissen empirisch analysiert und bewertet. Dazu wird jeweils die kontextbasierte Suche mit der herkömmlichen Textsuche verglichen. Im Vordergrund stehen dabei die Skalierbarkeit des Systems sowie die Auswirkungen des für die Arbeit zentralen Parameters zur Festlegung der Kontextgröße.

Abschnitt 5.1 stellt die beiden Szenarien vor, anhand derer die Messergebnisse ermittelt wurden. Die Qualität von Resultaten sowohl der Schlüsselwortsuche als auch der Suche nach ähnlichen Dokumenten wird in Abschnitt 5.2 analysiert. Die Abschnitte 5.3, 5.4 und 5.5 befassen sich mit der Messung und Bewertung des Ressourcenverbrauchs zur Erstellung des Index. Dabei wird sowohl die Größe des erstellten Index als auch die Laufzeit der Algorithmen analysiert. Die Laufzeit zur Bearbeitung von Suchanfragen sowohl der Schlüsselwortsuche als auch der Suche nach ähnlichen Dokumenten wird in Abschnitt 5.6 diskutiert. Abschnitt 5.7 fasst die präsentierten Messungen zusammen.

Kapitel 6: Zusammenfassung und Ausblick

Zum Abschluss der Arbeit enthält Kapitel 6 eine kurze Zusammenfassung des in dieser Arbeit vorgestellten Ansatzes sowie der Ergebnisse. In Abschnitt 6.2 wird das im Rahmen dieser Arbeit vorgestellte System in das in Abschnitt 2.5 eingeführte Kategorisierungsschema eingeordnet. Abschnitt 6.3 fasst die Abdeckung der Anforderungen zusammen. Schließlich werden in Abschnitt 6.4 offene Probleme angesprochen und weiterer Entwicklungs- bzw. Forschungsbedarf benannt.

2

Grundlagen

In diesem Kapitel werden wichtige Grundlagen sowie verwandte Arbeiten präsentiert und diskutiert. Jeder Abschnitt dieses Kapitels wird abgeschlossen durch eine Diskussion der jeweiligen Auswirkungen auf die Konzeption des im Rahmen der vorliegenden Arbeit vorgestellten Ansatzes sowie auf die Implementierung des Prototyps.

In Abschnitt 2.1 werden zunächst die für die vorliegende Arbeit grundlegenden Techniken aus dem Bereich des Information Retrieval vorgestellt. Die Abschnitte 2.2 und 2.3 stellen verwandte Arbeiten aus den Bereichen der Suche in relationalen Datenbanken bzw. in semistrukturierten Daten vor. Ansätze zur Integration strukturierter und unstrukturierter Daten sind in Abschnitt 2.4 zusammengefasst. Im Bereich der semantischen Suche existiert eine Vielzahl von verwandten Ansätzen. Abschnitt 2.5 gibt dazu eine kurze Einführung. Anschließend wird ein Kategorisierungsschema für semantische Suche präsentiert und eine Auswahl der analysierten Ansätze anhand dieses Schemas vorgestellt. In Abschnitt 2.6 wird die Einbettung des im Rahmen dieser Arbeit entwickelten Ansatzes ins betriebliche Wissensmanagement diskutiert. Abschnitt 2.7 präsentiert die Berechnung von kürzesten Wegen und Spreading Activation als Alternativen zur Auswertung von im ContextGraph abgelegter Information. In Abschnitt 2.8 werden schließlich die in diesem Kapitel präsentierten Inhalte kurz zusammengefasst.

2.1 Dokumentensuche

Mit der rasanten Entwicklung des Word Wide Web (kurz: Web) stieg die Bedeutung von Suchmaschinen. Heute existiert eine Vielzahl von kostenfrei zugänglichen

Suchmaschinen[1], welche es einem Benutzer erlauben, zu einer beliebigen Menge von Schlüsselwörtern Links auf Dokumente des Web zu finden, die von der jeweiligen Suchmaschine als relevant erachtet werden. Die Menge der Treffer und deren Bewertung sowie der Umfang der durchsuchbaren Dokumente unterscheiden sich dabei von Suchmaschine zu Suchmaschine.

Neben den populären Websuchmaschinen existieren Produkte, die sich auf das Durchsuchen von speziellen Teilen des Web, von Intranets, von Datenbanken oder sogar von einzelnen Computern spezialisieren.

Für den Rest der Arbeit bezeichne der Begriff *Schlüsselwortsuche* eine Suche, bei der der Benutzer den Informationsbedarf durch Schlüsselworte spezifiziert. Soll das Ergebnis des Suchprozesses hervorgehoben werden, so weist der Begriff *Dokumentensuche* darauf hin, dass jeweils gesamte Dokumente gefunden werden – im Gegensatz dazu werden bei der Suche in XML-Dokumenten Dokumententeile und bei der Suche in Datenbanken Tupel gefunden. Steht die Art der ausgewerteten Information im Vordergrund so werden die Begriffe *Textsuche*, *Volltextsuche* und *herkömmliche* Suche verwandt, um anzuzeigen, dass zur Beantwortung von Suchanfragen lediglich der Text der Dokumente ausgewertet wird. Demgegenüber wird der in dieser Arbeit verfolgte Ansatz zur Miteinbeziehung des Kontexts mit *kontextbasierter* Suche bezeichnet.

In der Informatik fällt die Thematik der Schlüsselwortsuche in den Bereich des umfassenderen Gebiets des Information Retrieval (IR). Dazu existiert eine unüberschaubare Menge an Veröffentlichungen, sowohl an Lehrbüchern als auch an wissenschaftlichen Artikeln. Gute Übersichten sind in den Lehrbüchern von Baeza-Yates und Ribeiro-Neto [1999]; Witten et al. [1999] und Grossman und Frieder [2004] oder im Überblicksartikel von Mitra und Chaudhuri [2000] zu finden. An dieser Stelle wird lediglich kurz auf die für die vorliegende Arbeit relevanten Konzepte eingegangen: Abschnitt 2.1.1 gibt einen Überblick über die Architektur von Suchmaschinen, Abschnitt 2.1.2 geht kurz auf den Aufbau von Indexen ein. Die Bewertung von Suchresultaten wird in Abschnitt 2.1.3 skizziert. Die Bewertung der Qualität von Suchmaschinen steht im Mittelpunkt von Abschnitt 2.1.4. Schließlich gibt Abschnitt 2.1.5 eine kurze Zusammenfassung der für die vorliegende Arbeit relevanten Aspekte.

2.1.1 Aufbau

Suchmaschinen für die Schlüsselwortsuche können aufgrund ihrer Architektur in zwei Kategorien aufgeteilt werden. Bei sogenannten *Online-Suchmaschinen* werden zur Zeit der Benutzeranfrage potenziell relevante Daten nach den angegebenen Schlüsselwörtern durchsucht und im Fall eines Suchtreffers dem Benutzer angezeigt. Da bei der Online-Suche zur Anfragezeit alle potenziell relevanten Daten vom Sy-

[1] z.B.: `www.google.com`, `http://www.ask.com/`, `http://seekport.de/`, `http://search.msn.com/`

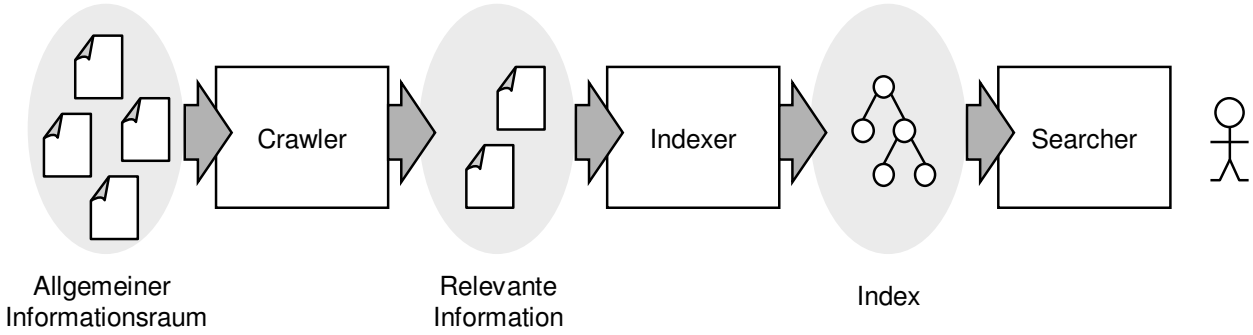


Abbildung 2.1: Informationsfluss in indexbasierten Suchmaschinen.

stem auf Übereinstimmung mit den angegebenen Schlüsselwörtern geprüft werden müssen, ist dieser Ansatz sehr zeitaufwändig. Die Online-Suche kommt daher für die interaktive Suche nur in Szenarien mit wenigen hundert Dokumenten in Frage.

Indexbasierte Suchmaschinen begegnen diesem Problem, indem die für den Benutzer potenziell interessanten Daten in Strukturen abgelegt werden, die einen effizienten Zugriff erlauben, die sog. *Indexe*. Somit orientieren sich die indexbasierten Suchmaschinen an dem in Abb. 2.1 dargestellten Informationsfluss. Der *Crawler* ist ein Werkzeug, welches aus einem allgemeinen Informationsraum diejenigen Daten auswählt, die von den Benutzern der Suchmaschine gefunden können werden sollen. Im Webszenario ist dies ein Programm, welches anhand der Verlinkungsstruktur des Web vollautomatisch von Webseite zu Webseite navigiert und für die Suchmaschine relevante Daten herunterlädt. Der *Indexer* bereitet anschließend die vom Crawler gefundenen Daten auf und stellt sie in einem Index dem *Searcher* zur Verfügung. Benutzeranfragen werden vom Searcher dann aufgrund der im Index abgelegten Daten bearbeitet. Sowohl Crawler als auch Indexer sind bei vielen Suchmaschinen als andauernd laufende Prozesse realisiert, die ständig den Index aktualisieren.

Auch bei kontinuierlicher Indexaktualisierung haben indexbasierte Suchmaschinen gegenüber Online-Suchmaschinen den Nachteil, dass zwischen den tatsächlichen Daten im allgemeinen Informationsraum und den im Index abgelegten Daten Inkonsistenzen auftreten können, was zu einer inkorrekten Beantwortung von Anfragen führen kann. Weiterhin ist anzumerken, dass die erhöhte Performanz der Anfrage durch einen erhöhten Speicherbedarf zur Ablage des Index erkauft wird.

2.1.2 Indexe

Der Begriff *Index* bezeichnet in der Informatik eine effiziente Zugriffsstruktur. Im Bereich der Schlüsselwortsuche wurden verschiedene Arten von Indexen entwickelt, die es einem System ermöglichen, eine effiziente Abbildung von Schlüsselwörtern auf entsprechende Dokumente oder Dokumententeile durchzuführen. Hier sind vor allem die weitverbreiteten *Inverted Files* bzw. Inverted Lists zu nennen, bei denen für jedes im Dokumentenstamm vorkommende Wort alle relevanten Dokumente ab-

gespeichert werden. Der Name Inverted File leitet sich von der Invertierung des Zugriffs ab. Während im normalen Gebrauch unter einem Dateinamen eine Menge von Wörtern abgelegt sind, werden bei Inverted Files jedem Wort eine Liste von Dokumenten zugeordnet. Zur Lokalisierung der Dokumentenlisten werden meist Baumstrukturen verwendet. Bei der Trie-Datenstruktur handelt es sich z.B. um einen Baum, bei dem die Kanten mit Buchstaben markiert sind. Eine Suche im Baum beispielsweise nach dem Wort "Klebenaht" beginnt dann bei der Wurzel und folgt von dort der mit "k" bezeichneten Kante und in der Folge den Kanten, die mit, "l", "e", "b", "e" , "n", "a", "h" und "t" bezeichnet sind. Das so erreichte Blatt des Baumes enthält einen Verweis auf die Liste aller Dokumente, in denen das Wort "Klebenaht" auftaucht. Neben Inverted Files sind Suffix Trees und Signature Files als alternative Indexstrukturen zu nennen. Da in dieser Arbeit lediglich auf existierende Indexstrukturen aufgebaut wird, sei für eine Einführung zum Thema Indexe auf [Baeza-Yates und Ribeiro-Neto, 1999, Kap. 8] und [Grossman und Frieder, 2004, Kap. 5] verwiesen.

2.1.3 Bewertung von Suchresultaten

Durch das rasante Anwachsen der Datenmengen ist ein reines Auflisten von Dokumenten, die die gesuchten Wörter enthalten nicht mehr ausreichend. Die Qualität moderner Suchmaschinen wird daher (neben der Performanz) entscheidend durch die Sortierung der Suchresultate und somit von deren Bewertung (engl.: Ranking) bestimmt. Für den Rest der Arbeit wird der Konvention gefolgt, Funktionen zur Bewertung von Suchresultaten als Bewertungs*maße* zu bezeichnen, obwohl sie mathematisch gesehen die Anforderungen an eine Maßfunktion nicht notwendigerweise erfüllen.

2.1.3.1 Informationsquellen

Bei der Bewertung von Dokumenten kann auf verschiedenste Informationsquellen zurückgegriffen werden. Die gängigsten Quellen lassen sich in folgende Klassen einteilen:

- Text: Die naheliegendste Informationsquelle zur Relevanzbewertung eines Dokuments ist der Text des Dokuments sowie Statistiken über den Textkorpus aller durchsuchbaren Dokumente.

- Metadaten: In vielen Szenarien sind zusätzlich zum Text eines Dokuments Daten vorhanden, die das Dokument charakterisieren, sog. Metadaten. Diese beinhalten typischerweise Autor, Datum und Speicherort des Dokuments. In manchen Fällen sind Schlüsselwörter, die den Inhalt des Dokuments beschreiben, als Metadaten verfügbar.

- Struktur des Textkorpus: In Anwendungsgebieten, in denen Dokumente untereinander verbunden sind, wie z.B. im Web durch Hyperlinks oder in Literaturdatenbanken durch Zitationen, kann die Struktur der Verbindungen zu einer Bewertung von Dokumenten beitragen. So können z.B. Dokumente einer Literaturdatenbank, die häufig zitiert werden, als relevant angenommen werden.

- Statistiken der Suchmaschine: Eine wichtige Informationsquelle wird oftmals von der Suchmaschine selbst erhoben. Dokumente werden dann als wichtiger erachtet, z.B. je häufiger sie abgerufen werden (sog. Click-Count) oder je länger sie betrachtet werden.

- Nutzerprofil und -rückmeldung: Vorangegangene Suchanfragen eines Nutzers bzw. einer Nutzergruppe und ggf. die benutzerseitige Bewertung der Resultate können herangezogen werden, um die Relevanz von Resultaten zu bewerten.

Bei der Bewertung von Dokumenten ist grundsätzlich zwischen der Wichtigkeit und der Relevanz von Dokumenten zu unterscheiden. Während die Relevanz eines Dokuments zur Anfragezeit dynamisch bezüglich einer Anfrage bewertet wird, legt die Wichtigkeit eines Dokuments ein anfrageunabhängiges Maß für die relative Wichtigkeit des Dokuments im Dokumentenstamm fest. Die Wichtigkeit kann somit statisch zur Indexierungszeit berechnet werden.

2.1.3.2 Relevanz eines Dokuments

Die Berechnung der Relevanz eines Dokuments bezüglich einer Suchanfrage bildet das Kernstück der Resultatsbewertung einer Suchmaschine. Ein Dokument wird dabei als umso relevanter eingestuft, je besser es den in der Suchanfrage spezifizierten Kriterien entspricht. Da die Relevanzbewertung erst zur Anfragezeit vorgenommen werden kann, spricht man auch von dynamischer Resultatsbewertung.

In diesem Bereich existiert eine Vielzahl von Veröffentlichungen, für Einführungen siehe z.B. [Baeza-Yates und Ribeiro-Neto, 1999, Kap. 2], [Grossman und Frieder, 2004, Kap. 2] oder [Singhal, 2001]. An dieser Stelle soll nur ein kurzer Überblick gegeben werden.

Die gängigsten Modelle zur Relevanz lassen sich in die folgenden Kategorien einteilen:

- Mengentheoretische Modelle. Das naheliegendste Modell zur Berechnung der Relevanz von Suchresultaten basiert auf der Mengenlehre und wird Boolesches Modell genannt. Ein Dokument wird genau dann als relevant bzgl. einer Anfrage erachtet, wenn es alle Suchbegriffe der Anfrage enthält. Auf diese Weise können Dokumente entweder als relevant oder als nicht-relevant eingestuft werden. Es existiert keine Abstufung der Relevanz.

- Algebraische oder Vektorraum-Modelle. In algebraischen Modellen werden sowohl Anfragen als auch Dokumente als Vektoren modelliert [Salton und Lesk, 1968]. Ihre Bekanntheit erlangten die Modelle als Teil des SMART-Systems [Salton, 1971]. Die Dimensionen des Vektorraums werden festgelegt durch die Anzahl der suchbaren Begriffe. Die Relevanz eines Dokuments bzgl. einer Anfrage wird aus dem Abstand der beiden zugehörigen Vektoren, z.B. per Kreuzprodukt, ermittelt. Das Boolesche Modell lässt sich als Spezialfall des Vektorraum-Modells modellieren, in dem die Vektoreinträge, die sog. Begriffsgewichte (engl.: Term Weights), auf die Werte "1" für: "Begriff kommt in Dokument/Anfrage vor" und "0" für: "Begriff kommt in Dokument/Anfrage nicht vor" beschränkt werden. Das Vektorraum-Modell lässt für die Begriffsgewichte jedoch Werte zu, die eine aussagekräftigere Bewertung ermöglichen. Üblicherweise wird zur Bestimmung der Gewichte auf das sog. $\mathrm{tf.idf}$-Maß zurückgegriffen. Das $\mathrm{tf.idf}$-Maß setzt sich aus einer tf-Komponente – für Term Frequency (Begriffshäufigkeit) – und einer idf-Komponente – für Inverse Document Frequency (Inverse Dokumentenhäufigkeit) – zusammen. Dabei steigt der tf-Wert mit der Anzahl der Begriffsvorkommnisse in einem Dokument. Der idf-Wert hingegen steigt mit der Seltenheit des Begriffs im gesamten Dokumentenstamm. Das $\mathrm{tf.idf}$-Maß erreicht somit, dass solche Dokumente als relevant bewertet werden, die eine möglichst große Anzahl der Begriffe der Suchanfrage enthalten, die, bezogen auf den Dokumentenstamm, selten auftreten. Auf das Vektorraum-Modell wird zusammen mit $\mathrm{tf.idf}$ in Kap. 3.2.1.2 weiter eingegangen.

- Probabilistische Modelle. Im Probabilistischen Modell basiert die Bewertung der Suchresultate auf der Bestimmung der Wahrscheinlichkeit, mit der ein Dokument für eine Anfrage relevant ist, siehe z.B. Spärck Jones et al. [2000]. Dabei wird der Prozess der Dokumentensuche als mehrstufiges Zufallsexperiment angesehen, bei dem der Benutzer durch Relevanz-Feedback der gefundenen Dokumente die Relevanz der Suchergebnisse erhöhen kann. Als Startwert für die Relevanz-Wahrscheinlichkeit eines Begriffs wird oft der idf-Wert des Begriffs verwendet. Die Probabilistischen Modelle gehen zurück auf Robertson und Jones [1976]. Das heute viel beachtete probabilistische Modell Okapi BM25 wurde von Robertson et al. [1998] beschrieben.

Zu allen genannten Modellen existiert eine Vielzahl von Modifikationen und Weiterentwicklungen, auf die an dieser Stelle nicht eingegangen werden kann.

2.1.3.3 Wichtigkeit eines Dokuments

Im Gegensatz zur Relevanz eines Dokuments wird dessen Wichtigkeit unabhängig von jeglicher Anfrage bestimmt. Die Wichtigkeit kann somit zur Indexierungszeit

berechnet werden und wird daher auch statische Resultatsbewertung genannt. Insbesondere für Suchmaschinen, welche eine großen Dokumentenstamm erfassen ist die Wichtigkeit von Dokumenten ein wesentliches Kriterium zur Bewertung von Suchresultaten, so beruht der Erfolg der Suchmaschine Google zumindest teilweise auf dem dort eingesetzten Wichtigkeitsmaß.

In die Berechnung der Wichtigkeit von Dokumenten können verschiedenste Faktoren einbezogen werden. Dabei ist zu beachten, dass diese Bewertung häufig für alle Benutzer der Suchmaschine gleichermaßen gilt. In wenigen Fällen ist es dem Benutzer möglich, zur Anfragezeit aus einer festgelegten Anzahl von statischen Bewertungsmodellen auszuwählen. Statische Bewertungsmodelle können unterschiedlichste Information beinhalten und können miteinander kombiniert werden. Beispiele für Modelle für die Wichtigkeit von Dokumenten sind:

- Änderungsdatum. In manchen Anwendungsgebieten ist es von Vorteil, wenn neuere Dokumente grundsätzlich besser bewertet werden als ältere. Eine derartige Funktionalität ist bspw. in Google Desktop[2] implementiert.

- Bezahlte Einträge. Bei manchen Web-Suchmaschinen kann eine Besserplatzierung von Webseiten gekauft werden. Andere fügen eine konstante Anzahl "kommerzieller" Links an der Spitze der Resultatsliste ein.

- Click-Count. Wie oben angesprochen, kann aus der Häufigkeit mit der dem Link auf ein bestimmtes Dokument gefolgt wird, auf dessen Beliebtheit bei Benutzern zurückgeschlossen werden. Click-Counts bergen jedoch folgendes Risiko: Wird ein Dokument auf Basis einer irreführenden Präsentation in der Ergebnisliste häufig fälschlicherweise angeklickt, so steigt automatisch dessen Click-Count. Dies wiederum führt zu einer besseren Bewertung und somit wiederum zur Erhöhung des Click-Count. Der Click-Count kann daher kaum als alleiniges Wichtigkeitsmaß eingesetzt werden.

- Struktur des Textkorpus. Die wohl populärste statische Bewertung, die die Struktur des Textkorpus ausnutzt, ist der in der Web-Suchmaschine Google verwendete PageRank [Brin und Page, 1998]. Dabei wird die Wichtigkeit einer Webseite darüber bestimmt, wie viele andere Seiten auf diese Seite verweisen und wie wichtig diese Seiten sind. Eine formale Definition von PageRank wird in Abschnitt 3.2.2.1 gegeben.

- Domänenspezifische Information. Neben den oben diskutierten statischen Bewertungsmodellen existiert eine Vielzahl an Ansätzen, die bestimmte Eigenschaften des Anwendungsbereichs beachten. Bei einer Intranetsuchmaschine könnten etwa Dokumente besser bewertet werden, die im Unternehmensbereich des Suchers erstellt wurden.

[2] Google Desktop, Version: 20051208-de, `http://desktop.google.com/` [20.12.2005]

2.1.3.4 Suche nach ähnlichen Dokumenten: What's Related

Streng genommen nicht in den Bereich der Schlüsselwortsuche fällt die Suche nach ähnlichen Dokumenten (engl.: Similarity Search, auch: What's Related). Dabei wird die Suchanfrage nicht durch eine Menge von Begriffen spezifiziert, sondern durch ein oder mehrere Dokumente.

Im Bereich des Information Retrieval findet sich eine Reihe von Ansätzen, die die Ähnlichkeit von Dokumenten über ihren textuellen Inhalt bestimmen. Dazu gehört unter anderem die Ähnlichkeitsberechnung zwischen Dokumenten, basierend auf dem Vektorraummodell und $\mathrm{tf.idf}$, wie in Abschnitt 3.2.1.2 beschrieben. Sollen zu einem Dokument ähnliche Dokumente gefunden werden, so wird aus dem Inhalt des Dokuments eine Anzahl von Wörtern extrahiert und diese dann als Teil einer Schlüsselwortanfrage an eine Suchmaschine abgesetzt.

Ursprünglich aus der Bibliometrie kommen Ansätze, die die Ähnlichkeit von Dokumenten nicht aufgrund ihrer textuellen Ähnlichkeit bewerten, sondern basierend auf ihrem Kontext, bzw. ihrer Verknüpfungsstruktur. In der Bibliometrie wird dazu auf die Verknüpfung von Dokumenten durch Zitate zurückgegriffen. In jüngerer Zeit finden sich derartige kontextbasierte Bewertungsmaße vor allem in Websuchmaschinen, die auf den Hyperlink-gestützten gegenseitigen Verweisen der Webseiten basieren.

Bei der kontextbasierten Ähnlichkeitsbewertung von Dokumenten werden grundsätzlich zwei Ideen verfolgt: Bei der bibliographischen Kopplung (engl.: Bibliographic Coupling) werden solche Dokumente als ähnlich betrachtet, welche auf viele gleiche Dokumente verweisen [Kessler, 1963]. Die Kozitation (engl.: Co-Citation) sieht hingegen zwei Dokumente als verwandt an, wenn viele Dokumente existieren, die jeweils beide zitieren [Small, 1973]. Beide Ansätze wurden verschiedentlich diskutiert und erweitert.

Im Bereich der Ähnlichkeitsbestimmung von Webseiten werden von Dean und Henzinger [1999] drei Algorithmen verglichen, die auf Kozitation und bibliographischer Kopplung basieren. Jeh und Widom [2002] schlagen vor, die Kozitation derart zu erweitern, dass ebenfalls solche Dokumente als ähnlich gelten, die ihrerseits von ähnlichen Dokumenten zitiert werden. Die Zitationsbeziehung zwischen Dokumenten wird somit als transitive Relation aufgefasst, anhand derer die Dokumentenähnlichkeit berechnet wird.

Im Bereich der frei zugänglichen Websuchmaschinen wird die Suche nach ähnlichen Dokumenten von mehreren Anbietern implementiert. Ihre Funktionsweise wird jedoch – wenn überhaupt – nur oberflächlich beschrieben, so z.B. bei Google[3], Netscape[4] oder Gigablast[5].

[3] "Find Similar"-Funktion: `http://www.google.com/help/features.html` [28.04.2006]

[4] "What's Related"-Funktion: `http://wp.netscape.com/escapes/related/faq.html` [28.04.2006]

[5] "Related Pages"-Funktion `http://www.gigablast.com/prrelated.html` [28.04.2006]

2.1.4 Bewertung der Qualität einer Suchmaschine

Die in den letzten beiden Abschnitten besprochene Bewertung der Suchresultate wird von der Suchmaschine durchgeführt und dient dazu, dem Benutzer die relevantesten Dokumente möglichst zu Beginn der Resultatsliste anzuzeigen. In diesem Abschnitt werden kurz die beiden wichtigsten Maße besprochen, die dazu verwendet werden, die Qualität einer Suchmaschine zu bewerten. Diese beiden Maße heißen Precision (etwa: Präzision) und Recall (etwa: Abruf oder Rücklauf) und werden folgendermaßen definiert [Baeza-Yates und Ribeiro-Neto, 1999; Witten et al., 1999]:

$$\text{Precision} = \frac{\text{Anzahl der gefundenen, relevanten Dokumente}}{\text{Anzahl der gefundenen Dokumente}}$$

$$\text{Recall} = \frac{\text{Anzahl der gefundenen, relevanten Dokumente}}{\text{Anzahl der relevanten Dokumente}}$$

Während Precision die Wahrscheinlichkeit ausdrückt, dass ein beliebiges gefundenes Dokument relevant ist, steht Recall für die Wahrscheinlichkeit, dass ein beliebiges, relevantes Dokument auch gefunden wird. Wird der Nenner einer der beiden Formeln 0 (Null), so wird das Ergebnis der jeweiligen Formel auf 1 (Eins) gesetzt.

Precision und Recall sind konkurrierende Maße, von denen jedes leicht auf Kosten des anderen bevorzugt werden kann: Eine triviale Suchmaschine, die stets die leere Menge findet, hat Precision=1 (alle gefundenen Dokumente sind relevant), dafür aber Recall=0. Andersherum hat eine Suchmaschine, die alle Dokumente des Dokumentenstamms findet, Recall=1, dafür jedoch eine kleine Precision. Auf dieser Beobachtung aufbauend, wurde eine Vielzahl von Qualitätsmaßen vorgeschlagen, die aus einer Kombination von Precision und Recall bestehen [Witten et al., 1999; Baeza-Yates und Ribeiro-Neto, 1999]. Allen gemeinsam ist die Ausgangslage, dass zur Bestimmung der Qualität einer Suchmaschine die Relevanz der gefundenen Dokumente bzgl. einer Suchanfrage jeweils von einem menschlichen Benutzer beurteilt werden muss. Eine vollautomatische Bewertung der Qualität ist somit unmöglich. Im Bereich der Volltextsuche hat sich auf dieser Basis die TREC-Initiative das Ziel gesetzt, ein Forum zum qualitativen Vergleich von Suchmaschinen zu schaffen [TREC, 2006].

2.1.5 Bedeutung für den Entwurf

Die hier vorliegende Arbeit hat zum Ziel, im Bereich Information Retrieval erfolgreich erprobte Technologien und Algorithmen auf den Bereich der kontextbasierten Dokumentensuche im Intranet zu übertragen. Wie das Information Retrieval im allgemeinen verbessert werden kann, steht somit nicht im Vordergrund. Bedeutung für die vorliegende Arbeit haben daher allgemeine Erkenntnisse zum Aufbau von Suchmaschinen, Technologien zur Indexerstellung und -verwaltung und die Bewertung von Suchresultaten. Jeder dieser Bereiche wird in Kapitel 3 auf seine Tauglichkeit für

bzw. seine Übertragbarkeit auf die Intranetsuche untersucht. So wird in Abschnitt 3.2 das Vektorraummodell und tf.idf auf das kontextbasierte Szenario übertragen. Zur Bestimmung der kontextbasierten Wichtigkeit wird ein vom PageRank abgeleitetes Verfahren zur Bewertung von Datenbankgraphen eingesetzt. Für die Suche nach ähnlichen Dokumenten wird ein Bewertungsmaß eingeführt, welches als Erweiterung von Kozitation und bibliographischer Kopplung gesehen werden kann. Der Abschnitt zur Bewertung der Qualität von Suchmaschinen bildet schließlich die Basis zur Evaluation des Prototyps, wie in Abschnitt 5.2 diskutiert.

2.2 Schlüsselwortsuche in relationalen Datenbanken

Im Datenbankbereich existieren verwandte Arbeiten zu einem Thema, welches "database exploration" (etwa: "Datenbankerkundung") genannt wird. Ein typisches Anwendungsszenario besteht darin, die Inhalte der Datenbank für eine Volltextsuche zu erschließen. Sendet ein Benutzer eine Schlüsselwortanfrage an das System, so erhält er ein oder mehrere Ergebnisobjekte (im relationalen Fall besteht ein Ergebnisobjekt aus einem Tupel), die jedes nach Möglichkeit sämtliche Begriffe der Anfrage enthalten. Als Anfrage werden lediglich einzelne Suchbegriffe angegeben, die ggf. mit den booleschen Operatoren UND, ODER und NICHT verknüpft sind. Der Unterschied zu gewöhnlichen, strukturierten Datenbankanfragen (etwa mittels SQL) besteht darin, dass einerseits dem Benutzer oder der anfragenden Anwendung die Struktur der Datenbank nicht bekannt sein muss. Andererseits sind – wie bei der Schlüsselwortsuche in Dokumenten – unscharfe Anfragen, z.B. nach Teilen von Wörtern, möglich.

Im Bereich der *Relationalen Datenbanken* (RDBMS) liefert die Schlüsselwortsuche eine Menge von Tupeln zurück, von denen jedes möglichst alle Suchbegriffe enthält. Um solche Tupel zu erzeugen, müssen häufig mehrere Tabellen verbunden werden. Die Systeme unterscheiden sich einerseits hinsichtlich der Mechanismen zum Verbund von Tabellen. Andererseits werden bei der Bewertung und somit der Sortierung der Suchergebnisse verschiedene Ansätze verfolgt.

Die Ansätze haben gemeinsam, dass sie zur performanten Bearbeitung der Suchanfragen Indexe verwenden, in denen die suchbaren Begriffe redundant zur Datenbank abgelegt werden. Der naheliegende Ansatz dabei ist, die Daten jeweils dem Tupel zuzuordnen, in denen sie zu finden sind. Bei einer Suche nach "Klebenaht" würden z.B. die Primärschlüssel aller Tupel gefunden, die das Wort "Klebenaht" in einem Attribut enthalten, also etwa den Primärschlüssel F18 des folgenden Tupels einer Tabelle db.Fehlerursachen: `db.Fehlerursachen<F18, 'Klebenaht', ...>`. Abweichend davon wird in [Agrawal et al., 2002] ein Index gröberer Granularität untersucht. Die suchbaren Begriffe werden dazu nicht ihren jeweiligen Tupel-Identifikatoren zugeordnet, sondern lediglich ihren Tabellen- und Tabellenattributs-Identifikatoren. Die Suche nach "Klebenaht" ergäbe somit lediglich das Ergebnis `db.Fehlerursachen.Ort`, falls das Wort "Klebenaht" in einem Attribut mit der

Bezeichnung “Ort” abgelegt ist. Während des Suchvorgangs muss dann jeweils eine SELECT-Anfrage an die Datenbank abgesetzt werden, die das entsprechende Tupel zurückliefert, also etwa: `SELECT * FROM db.Fehlerursachen WHERE ort LIKE '%Klebenaht%'`.

Ansätze, die Datenbanken zur Implementierung der Indexe verwenden, wie etwa in Chaudhuri et al. [2005], werden im Folgenden nicht diskutiert. Bei der Vorstellung der wichtigsten Ansätze wird in chronologischer Reihenfolge vorgegangen. Eine Einführung in v.a. ältere Ansätze ist bei [Grossman und Frieder, 2004, Kap. 6.2] zu finden.

Im Folgenden wird auf die Ansätze DataSpot, Jungle, BANKS, DBXplorer und DISCOVER, sowie auf einige am Markt verfügbare Produkte eingegangen.

2.2.1 DataSpot

Das DataSpot System [Dar et al., 1998; Palmon, 1998; Palmon und Geva, 1998] integriert heterogene Datenbanken in einem graphenbasierten Modell, der sog. Hyperbase. Die Daten der Hyperbase werden per Schlüsselwortsuche zugänglich gemacht.

Eine Hyperbase ist eine Graphenstruktur, deren Knoten entweder anonyme, innere Knoten oder mit Daten behaftete Blattknoten sein können. Die Kanten der Hyperbase werden in zwei Kategorien aufgeteilt: “Simple Edges” (Einfache Kanten) bilden einen gerichteten, azyklischen Graphen und modellieren somit hierarchische Zusammenhänge. “Identification Edges” (etwa: Benennungskanten) modellieren die Querverweise im Graphen. Bei der Abbildung einer relationalen Datenbank auf eine Hyperbase werden Tabellen, Tupel und Attribute auf innere Knoten abgebildet, die durch Einfache Kanten verbunden sind. Die Werte in den Attributen werden durch Blattknoten repräsentiert. Fremdschlüsselbeziehungen zwischen Tupeln werden auf Benennungskanten abgebildet.

Das Ergebnis einer Suchanfrage besteht bei DataSpot aus einer Liste von Hyperbase-Teilgraphen. Für die Bewertung der Resultate werden verschiedene Vorschläge gemacht, in die die Anzahl der Knoten, die Anzahl der Kanten, und die Anzahl der Kanten gleichen Typs einfließen. Alle vorgeschlagenen Bewertungsmaße greifen jedoch nur auf die Struktur des Graphen zurück, Worthäufigkeiten werden nicht miteinbezogen.

Ob das DataSpot-System zur Beschleunigung der Suchanfragen Indexe verwendet, konnte nicht geklärt werden.

2.2.2 Jungle

Der Jungle-Ansatz zur Erkundung von relationalen Datenbanken wurde von Böhlen et al. [1999] vorgestellt. Die Jungle-Suchmaschine ist in der Lage, eine Schlüsselwortsuche auf den Daten und Metadaten einer Datenbank durchzuführen. Die Anfragesprache AQUA unterscheidet zwischen “vertikalen” und “horizontalen” Anfra-

gen. Vertikale Anfragen betreffen die Metadaten einer Datenbank und sind in der Lage, Datenbanken, Tabellen, oder Tabellenattribute zu finden. Horizontale Anfragen betreffen die Daten der Datenbank. Laut [Böhlen et al., 1999] ist Jungle in der Lage, für Anfragen mit mehreren Schlüsselwörtern durch Fremdschlüsselbeziehungen verknüpfte Tabellen so zu verbinden, dass Tupel entstehen, die alle Schlüsselworte enthalten. Zur Realisierung dieser Funktionalität nennt die Veröffentlichung jedoch keine Details. Ebensowenig wird die Bewertung von Suchresultaten diskutiert.

2.2.3 BANKS

Das BANKS System [Bhalotia et al., 2002; Hulgeri et al., 2001] realisiert die Volltextsuche in RDBMS, indem Datenbankdaten als Graph repräsentiert werden. Dabei bilden die Tupel der Datenbank die Knoten des Datenbankgraphen, Fremdschlüsselbeziehungen zwischen Tupeln werden auf Kanten abgebildet. Das Ergebnis von Schlüsselwortanfragen besteht bei BANKS aus einer Menge von (zusammenhängenden) Subgraphen des Datenbankgraphen, sog. Steiner-Bäumen. Dabei beinhaltet jeder Subgraph für jeden Begriff der Anfrage mindestens einen Knoten, in dem dieser Begriff vorkommt.

Im Datenbankgraph von BANKS wird große inhaltliche Ähnlichkeit durch ein kleines Kantengewicht repräsentiert. Die Kanten sind gerichtet und Kanten treten stets in Paaren auf, sodass zwei Knoten u und v immer durch zwei Kanten $\langle u, v\rangle$ und $\langle v, u\rangle$ verbunden sind. Kantengewichte werden für alle Kanten, die in Richtung der Fremdschlüsselbeziehung gerichtet sind, von Administratoren vergeben. Die Gewichte der inversen Kanten werden automatisch berechnet. Diese automatische Berechnung basiert auf der Annahme, dass eine Kante umso weniger inhaltliche Ähnlichkeit repräsentiert, desto mehr Kanten des gleichen Typs von ihrem Quellknoten ausgehen. Dies wird durch folgendes Beispiel veranschaulicht: In einer Organisation sind alle Mitarbeiter durch eine Kante des Datenbankgraphen mit ihrer jeweiligen Abteilung verbunden. Somit sind ebenfalls Mitarbeiter derselben Abteilung transitiv miteinander verbunden. Bei Abteilungen mit wenigen Mitarbeitern ist die Wahrscheinlichkeit einer inhaltlichen Verbindung zwischen zwei Mitarbeitern höher, als bei Abteilungen mit vielen Mitarbeitern. Diese Situation wird dadurch modelliert, dass die Mitarbeiter-Kanten, ausgehend von einem Abteilungsknoten, der eine größere Abteilung repräsentiert und somit viele Mitarbeiter-Kanten hat, weniger inhaltliche Ähnlichkeit kodiert, als Mitarbeiter-Kanten einer kleineren Abteilung. Die Bewertung der Knoten ist inspiriert durch den in Abschnitt 2.1.3.3 angesprochenen PageRank-Algorithmus. In BANKS werden die Knoten des Datenbankgraphen entsprechend der Anzahl ihrer eingehenden Kanten (sog. In-Degree) bewertet.

Die Bewertung und somit die Sortierung der Subgraphen eines Suchergebnisses basiert ausschließlich auf einer Bewertung der Struktur des Subgraphen. Dazu werden sowohl die Gewichte der enthaltenen Knoten als auch die der enthaltenen Kanten berücksichtigt. Die textuellen Inhalte der Knoten gehen nicht in die Bewertung ein,

so spielt es z.B. keine Rolle, wie oft ein gesuchter Begriff in einem Knoten auftaucht oder wie oft er im gesamten Datenbestand enthalten ist. Eine Bewertung wie aus dem IR-Bereich findet somit nicht statt. Die Bewertung eines Resultat-Subgraphen setzt sich dann aus der Summe der Knotengewichte und aus dem Kehrwert der Summe der Kantengewichte des Subgraphen zusammen. Dadurch wird erreicht, dass Subgraphen, die viele wichtige Knoten haben, bevorzugt werden. Gleichzeitig werden – beeinflusst durch den Kehrwert der Summe der Kantengewichte – kleine Subgraphen mit kleinen Kantengewichten besser bewertet.

Das BANKS-System verwendet zur Bearbeitung von Suchanfragen einen Index, der die Zuordnung aller suchbaren Begriffe zu den Knoten des Datenbankgraphen leistet. Somit kann für eine Suchanfrage $S = s_1, \ldots, s_n$ mit Schlüsselwörtern s_1 bis s_n effizient eine Knotenmenge $K = K_1 \cup \cdots \cup K_n$ bestimmt werden, bei der gilt, dass jeder Knoten der Menge K_i das Schlüsselwort s_i (für $i = 1, \ldots, n$) enthält. Sodann wird auf allen Knoten $k \in K$ parallel ein kürzeste-Wege-Algorithmus ausgeführt, der die Nachbarschaft von k bestimmt. Diese Nachbarschaften werden solange expandiert bis eine Menge von Knoten $\{k_1 \in K_1, \ldots, k_n \in K_n\}$ gefunden wird, deren Nachbarschaften sich zumindest paarweise überlappen, d.h. es entsteht ein zusammenhängender Subgraph, der alle gesuchten Begriffe enthält. Dieser Subgraph wird anschließend bewertet und in die Ergebnismenge aufgenommen. Die Suchbegriffe der Anfragen in BANKS sind stets implizit untereinander mit UND verknüpft. Eine Analyse der Laufzeit von Suchanfragen ist in [Bhalotia et al., 2002] nicht zu finden.

2.2.4 DBXplorer und DISCOVER

DBXplorer und DISCOVER sind zwei ähnliche Ansätze, die in [Agrawal et al., 2002], bzw. [Hristidis und Papakonstantinou, 2002; Hristidis et al., 2003a] vorgestellt werden. Ähnlich wie die horizontalen Anfragen bei Jungle fokussieren beide auf die Beantwortung von Schlüsselwortanfragen auf den Daten einer Datenbank. Bei DBXplorer und DISCOVER steht die Generierung optimaler Tabellen-Verbunde im Vordergrund. In beiden Ansätzen wird zur Bearbeitung einer Schlüsselwortanfrage ein Index verwendet, um die Tabellen, Attribute und Tupel zu finden, die mindestens ein Schlüsselwort enthalten. Darauf aufbauend wird eine Menge von Join-Trees (DBXplorer) bzw. Joining-Networks (DISCOVER) generiert. Jedes Element dieser Mengen beschreibt, welche Tabellen der Datenbank verbunden werden müssen, um Tupel erzeugen zu können, die alle Suchbegriffe enthalten. Gemäß dieser Beschreibungen werden schließlich SQL-Anfragen generiert und an das zugrundeliegende Datenbanksystem abgesetzt.

Die Bewertung der resultierenden Tupel findet bei Agrawal et al. [2002] und bei Hristidis und Papakonstantinou [2002] ausschließlich basierend auf der Anzahl der nötigen Verbunde statt. Je weniger Tabellen-Verbunde zur Generierung eines Tupels notwendig sind, desto besser wird ein Tupel bewertet. Dies geschieht vor dem Hintergrund, dass Tupel, die viele Tabellenverbunde erfordern, oftmals weniger exakt

zur angegebenen Schlüsselwortanfrage passen als Tupel, die keine oder nur wenige Verbunde benötigen. Bei Hristidis et al. [2003a] fließt zusätzlich zur Anzahl der nötigen Verbunde eine tf.idf-Bewertung der gefundenen Tupel in die Bewertung mit ein. Weder DBXplorer noch DISCOVER bietet die Möglichkeit, ähnlich wie bei BANKS einzelne Verbunde mit unterschiedlichen Gewichtungen zu bewerten. Eine Erweiterung der Konzepte um diese einfache Möglichkeit, domänenspezifisch Einfluss auf die Bewertung der Suchergebnisse zu nehmen, sollte jedoch leicht möglich sein.

2.2.5 Am Markt verfügbare Produkte

Neben einer Reihe von Lösungen kleinerer Anbieter, bei denen datenbankexterne Suchmaschinen den Inhalt von Datenbanken indizieren und durchsuchen, existieren in den Produkten der drei derzeit am stärksten am Markt vertretenen Anbieter – Oracle, Microsoft und IBM – ebenfalls Ansätze zur Textsuche in Datenbanken.

Alle drei haben gemeinsam, dass in Datenbanktabellen und ggf. Dokumenten lediglich *isoliert* gesucht werden kann. Fremdschlüsselbeziehungen, wie sie in [Bhalotia et al., 2002] zur Erstellung des Datenbankgraphen oder in [Agrawal et al., 2002; Hristidis und Papakonstantinou, 2002; Hristidis et al., 2003a] zur Erzeugung möglichst passender Tupel verwendet werden, werden von keiner der drei Lösungen ausgenutzt.

Oracle bietet mit dem Oracle Ultra Search [Oracle, 2004, 2006] eine Suchmaschine, die auf Oracles Dokumentensuchmaschine Oracle Text Search [Oracle, 2005] basiert. Dementsprechend können nicht nur Datenbanktabellen, sondern beliebige Dokumentenbestände, wie z.B. Webseiten, Emails oder der Inhalt von Content-Management-Systemen von der Suche erfasst werden. Das Ranking von Ultra Search basiert auf dem Information-Retrieval-orientierten Ranking von Oracle Text Search.

Die Suchfunktionalität von Microsofts SQL Server 2005 Full-Text Search [Cencini, 2006] fokussiert mehr auf die eigentliche Erkundung von Datenbanken und beschränkt sich auf das Indexieren und Durchsuchen von Datenbanktabellen sowie von Dokumenten, die als Text in Tabellen abgelegt sind. Je nach Anfrageart wird ein anderer Rankingmechanismus angewendet, der sich ähnlich zu Ultra Search an den im Bereich des Information Retrieval entwickelten Maßen orientiert.

Der DB2 Net Search Extender von IBM [IBM, 2004, 2006] erfasst sowohl Datenbanktabellen (auch föderierter Datenbanken) als auch mittels IBM's Data Link Technologie in die Datenbank importierte Dokumente. Eine Textsuche, welche gleichzeitig die Text der Dokumente sowie deren Metadaten oder deren weiteren Kontext erfasst, wird nicht angeboten. Über die Art der Bewertung der Resultate konnte keine nähere Information gefunden werden.

2.2.6 Bedeutung für den Entwurf

Das Anwendungsgebiet der hier vorgestellten Ansätze zur Schlüsselwortsuche in RDBMS unterscheidet sich von der Schlüsselwortsuche im Intranet sowohl in der Art der zu durchsuchenden Daten als auch im Ziel der Recherche.

Anders als bei der Erkundung von Datenbanken sind bei der Intranetsuche Dokumente von Interesse, die oft mehr Begriffe umfassen als der Inhalt von Tupeln. Es liegt daher nahe, den Text der Dokumente mit Methoden aus dem Bereich des Information Retrieval zu behandeln. Dies betrifft insbesondere die Bewertung der Relevanz von Suchresultaten. Keiner der vorgestellten Ansätze berücksichtigt diesen Aspekt.

Die hier vorgestellten Ansätze verfolgen unterschiedliche Strategien bzgl. der Einbindung der Datenbank in die Suche. Jungle, DBXplorer und DISCOVER verbinden die Tupel der Datenbank erst zur Anfragezeit. Dies impliziert, dass die Suche teilweise online direkt auf der eigentlichen Datenbank durchgeführt wird. Zur Erkundung von Datenbanken ist dieser Ansatz sicherlich sinnvoll, da so stets mit der Datenbank konsistente Resultate erzeugt werden. BANKS und DataSpot gehen hier einen anderen Weg. Ihre Strategie besteht darin, die zu durchsuchenden Daten zur Indexierungszeit einmalig zu erheben und zu indexieren und zur Anfragezeit auf Datenbankzugriffe zu verzichten. Für eine wie in dieser Arbeit vorgestellte Intranetsuchmaschine, die potenziell von mehreren Benutzern gleichzeitig verwendet wird, muss der Offline-Ansatz von BANKS und DataSpot als überlegen angesehen werden. Im Offline-Ansatz sind die Suchergebnisse zwar nicht zu jeder Zeit konsistent zu den in der Datenbank abgelegten Daten. Jedoch profitiert die Performanz sowohl der Datenbank als auch der Suchmaschine von dieser Trennung. Im Szenario der Dokumentenrecherche, in dem gefundene Information nicht im Sekundenbereich aktuell sein muss, ist somit der Offline-Ansatz vorzuziehen.

Der in der vorliegenden Arbeit zur Modellierung von Dokumentenkontext eingesetzte ContextGraph (siehe Abschnitt 3.1) entspricht dem Datenbankgraph bei BANKS bzw. der Hyperbase bei DataSpot. Alle haben einerseits die Abstraktion von Datenbanken und andererseits die graphenbasierte Vernetzung der Information zum Ziel. Ähnlich zum Datenbankgraphen bei BANKS kann eine Kanten des ContextGraph mit einem Maß für die inhaltliche Ähnlichkeit der beiden durch die Kante verbundenen Knoten annotiert werden. Bei der Auswertung der im Graphen abgelegten Information unterscheidet sich die vorliegende Arbeit jedoch fundamental. Dies betrifft nicht nur die oben angesprochene Analyse der Texte in den Knoten, sondern auch die Strategie zur Auswertung der Verknüpfungsinformation. Während bei BANKS zur Anfragezeit mehrfach ein kürzeste-Wege-Algorithmus ausgeführt werden muss, wird in der vorliegenden Arbeit die Verknüpfungsinformation nur einmalig zur Indexierungszeit ausgewertet.

2.3 Suche in semistrukturierten Daten

Mit der zunehmenden Verbreitung des XML-Datenformats entstand die Notwendigkeit zur Schlüsselwortsuche in semistrukturierten Daten. Im Gegensatz zu strukturierten Daten, bei denen eine strikte Trennung zwischen Daten und Metadaten vorliegt, ist diese bei semistrukturierten Daten aufgehoben [Amer-Yahia und Shanmugasundaram, 2005; Bry et al., 2001; Abiteboul et al., 2000]. Die gängigen Beispiele für unstrukturierte, semistrukturierte und strukturierte Daten sind daher Textblöcke, XML-Dokumente und relationale Datenbanksysteme. Als Pendant zur *Text Retrieval Conference* TREC [TREC, 2006] entstand im Bereich der XML-Suche die *Initiative for the Evaluation of XML Retrieval* (INEX) [INEX, 2006]. Sowohl TREC als auch INEX verfolgen das Ziel, die Qualität von Suchmaschinen an vorgegebenen Datensätzen zu quantifizieren.

Für die vorliegende Arbeit ist die Suche in semistrukturierten Daten nicht direkt relevant. Tatsächlich lassen sich jedoch einige Ideen aus diesem Bereich auf die Suche im Intranet übertragen.

Wie bei relationalen Datenbanken gibt es auch im XML-Bereich grundsätzlich zwei verschiedene Arten von Suchen.

- Strukturierte Anfragen. Die Bearbeitung von strukturierten Anfragen auf XML-Dokumenten hat ihre Ursprünge im (XML-) Datenbankbereich. Im Mittelpunkt steht dabei die Auswertung der (weitgehend hierarchischen) Dokumentstruktur von XML-Dokumenten. Die Anfragen enthalten in der Regel Konstrukte, die die Dokumentstruktur mit Hilfe sog. Pfadausdrücke (Path Expressions) erschließen. Ein Beispiel für Pfadausdrücke im XML-Bereich findet sich in [W3C, 1999], aber auch viele andere semistrukturierte Datenmodelle verfügen über dieses Konzept, vgl. etwa [Abiteboul et al., 1997]. Eine populäre strukturierte XML-Anfragesprache ist das vom W3C entwickelte XQuery W3C [2006]. Andere Ansätze erweitern strukturierte Anfragesprachen um Elemente aus dem Information Retrieval [Amer-Yahia et al., 2005; Kaushik et al., 2004; Al-Khalifa et al., 2003]. Bei der Verarbeitung der XML-Suchanfragen steht im Vordergrund, dass das Ergebnis eines Pfadausdrucks aus einer Menge möglicherweise geschachtelter XML-Elemente besteht. Anders als bei strukturierten Anfragen an RDBMS, bei denen zur Anfragezeit die genaue Struktur der Ergebnisse bekannt ist, können daher zwei Ergebnisse derselben strukturierten XML-Anfrage völlig unterschiedliche interne Strukturen aufweisen.

- Unstrukturierte Anfragen. Ansätze zur Bearbeitung von unstrukturierten bzw. Schlüsselwortanfragen, wie z.B. [Guo et al., 2003; Hristidis et al., 2003b; Schmidt et al., 2001], sind mit Textsuchmaschinen vergleichbar. Während bei herkömmlichen Textsuchmaschinen stets gesamte Dokumente gefunden werden, kann die Rückgabe einer XML-Suchmaschine wie bei strukturierten An-

fragen aus einem oder mehreren potenziell geschachtelten Elementen oder sogar aus Attributen bestehen. Dies wirft für XML-Suchmaschinen das Problem auf, dass – abgesehen von der Hierarchiebeziehung zwischen Elementen – a priori nicht klar ist, welche Daten eines Dokuments als inhaltlich zusammenhängend gewertet werden können. Ein wichtiger Teil der Suche besteht daher darin, den Umfang jedes einzelnen Resultats zu bestimmen.

Da strukturierte Anfragen stets in einer formalen Sprache definiert werden müssen und somit nicht der in Abschnitt 1.2 formulierten Anforderung (1) nach Schlüsselwortsuche genügen, liegt der Fokus der folgenden Betrachtungen auf der Verarbeitung unstrukturierter Anfragen. In jüngerer Zeit entstand eine große Anzahl von Arbeiten im Bereich XML-Retrieval. An dieser Stelle kann kein umfassender Überblick über alle Arbeiten gegeben werden, vielmehr sollen diejenigen Ansätze vorgestellt werden, die einen direkten Zusammenhang mit den in dieser Arbeit vorgestellten Ideen aufweisen. Dazu zählen insbesondere die beiden im Folgenden dargestellten Ansätze, welche XML-Dokumente als Graphen modellieren.

2.3.1 Schlüsselwortsuche auf einer Lorel-Datenbank

Noch vor der Verbreitung von XML-Dokumenten wurden von Goldman et al. [1998] die Konzepte der Volltextsuche auf die Daten einer (semistrukturierten) Lorel-Datenbank [Quass et al., 1995] angewandt. In einer Lorel-Datenbank werden Daten gemäß dem Object-Exchange-Model (OEM) -Datenmodell [Papakonstantinou et al., 1995] abgelegt. Das OEM-Datenmodell stellt ein frühes Modell für semistrukturierte Daten dar, in welchem Daten und Metadaten in Form eines Graphen modelliert werden. Die Blätter des Graphen enthalten die Daten, während die inneren Knoten bzw. Kanten die Verknüpfungsinformation sowie die Metadaten kodieren. Der von Goldman et al. [1998] vorgestellte Ansatz annotiert OEM-Graphen mit Kantengewichten, die den inhaltlichen Zusammenhang zwischen Daten modellieren. Bei der Volltextsuche werden vom Benutzer zwei Mengen von Schlüsselwörtern definiert, die *find* und *near* genannt werden. Die vom System zurückgelieferte Ergebnismenge enthält OEM-Objekte, die Schlüsselwörter der find-Menge enthalten und einen möglichst kleinen Abstand von Objekten haben, die Schlüsselwörter der near-Menge enthalten. Der Abstand zwischen Objekten wird dabei durch die Kantengewichte des OEM-Graphen bestimmt.

2.3.2 SphereSearch

Die XML-Suchmaschine SphereSearch [Graupmann et al., 2005] realisiert zwar eine strukturierte Anfragesprache zur Suche auf XML-Dokumenten. In Bezug auf die vorliegende Arbeit ist jedoch vor allem die Modellierung der zu durchsuchenden Dokumente als Graph von Interesse. XML-Elemente werden in diesem Graphen als

Knoten repräsentiert. Die hierarchische Beziehung zwischen Elementen sowie nicht-hierarchische Beziehungen – wie z.B. durch XLink realisiert – werden auf Kanten abgebildet. Das Ergebnis einer Suche besteht bei SphereSearch aus einer Liste von Teilgraphen. Jeder Teilgraph erfüllt dabei die in der Anfrage formulierten Bedingungen, die sowohl den Inhalt der Knoten als auch deren Verknüpfung betreffen kann. Die Bewertung eines Teilgraphen basiert dabei auf einer Kompaktheits-Analyse desselben. Kleine Teilgraphen, bei denen die Pfade zwischen allen erforderlichen Knoten kurz sind, werden dabei vorgezogen, was, wie in Abschnitt 2.2.3 diskutiert, der Bewertung der Subgraphen bei BANKS entspricht. Um bei der Schlüsselwortsuche die Relevanz eines einzelnen Knoten zu bewerten, werden die umliegenden Graphknoten analysiert und entsprechend ihres Abstands in Bereiche (sog. Spheres) eingeteilt. Ein Knoten wird dann als um so relevanter bewertet, je mehr Schlüsselworttreffer in seiner unmittelbaren Nachbarschaft zu finden sind.

Wie bei der Schlüsselwortsuche in RDBMS verwendet die Implementierung des SphereSearch-Systems Indexe für schnellen Zugriff auf suchbaren Text. Weiterhin wird zur Beschleunigung der Kompaktheitsanalyse Pfadinformation im Index hinterlegt. Die Berechnung der Spheres wird jedoch zur Anfragezeit vorgenommen.

2.3.3 Bedeutung für den Entwurf

Die beiden hier vorgestellten Ansätze zur Schlüsselwortsuche in semistrukturierten Daten unterscheiden sich sowohl hinsichtlich ihres Fokus als auch bzgl. ihrer Realisierung. Für die vorliegende Arbeit sind sie insofern interessant, als sie auf einer Graphenrepräsentation der Daten sowie auf einer Kürzesten-Wege-Analyse zwischen Knoten des Graphen basieren. Im Gegensatz zu diesen beiden Ansätzen unterscheidet sich das dieser Arbeit zugrundeliegende Szenario darin, dass nicht Subgraphen des Datengraphen gefunden werden sollen, sondern lediglich einzelne, dedizierte Knoten. Diese Einschränkung rührt daher, dass, anders als im XML- und im Lorel-Szenario, zur Indizierungszeit der Umfang der suchbaren Dokumente feststeht. Dies führt dazu, dass Ranking-Maße aus dem Information Retrieval leichter auf die Intranet-Suche übertragen werden können.

Verglichen mit dem Ansatz der Find-Near-Mengen bei der Schlüsselwortsuche auf der Lorel-Datenbank, wird in der vorliegenden Arbeit ein allgemeineres Konzept verfolgt, welches den Find-Near-Ansatz als Spezialfall beinhaltet. Details hierzu finden sich in Abschnitt 4.6.2.

Der Sphere-Search-Ansatz kann mit der vorliegenden Arbeit insofern als verwandt angesehen werden, als Texte sowie Metadaten als Knoten ein und desselben Graphen aufgefasst werden. Unterschiede ergeben sich jedoch in der Berechnung und der Bewertung der Suchresultate. Einerseits werden bei SphereSearch Teile der Verknüpfungsinformation erst zur Anfragezeit ausgewertet. Im Konzept der vorliegenden Arbeit wird diese Berechnung im Rahmen einer Kürzesten-Wege-Analyse vollständig zur Indexierungszeit vorgenommen, siehe Abschnitt 4.1. Andererseits

sieht SphereSearch keine Kantengewichtung vor und verfügt über kein Konzept, Knoten, welche in vielen Kontexten auftreten, weniger stark zu gewichten als andere. Die Diskussion zur Bewertung von Suchresultaten für die vorliegende Arbeit findet sich in Abschnitt 3.2.

2.4 Integration strukturierter und unstrukturierter Daten für die Suche

In diesem Abschnitt werden Ansätze vorgestellt, die sich auf die Integration strukturierter und unstrukturierter Daten konzentrieren. Während die vorliegende Arbeit strukturierte Daten für die Verarbeitung unstrukturierter Anfragen einsetzt, gehen die im folgenden diskutierten Ansätze den umgekehrten Weg und verwerten die Ergebnisse von Textsuchmaschinen bei der Verarbeitung von SQL-Anfragen.

2.4.1 Kopplung von Datenbank und Suchmaschine

Der von Dessloch und Mattos [1997] vorgestellte Ansatz beschreibt ein Szenario, in dem ein Datenbanksystem direkten Zugriff auf den Inhalt von Textdokumenten hat, wie z.B. in einem DBMS-gestützten Content Management System. Um diese Texte für den Datenbankanwender zu erschließen, werden sie einer Textsuchmaschine zugänglich gemacht, die mittels SQL-Befehlen aufgerufen wird und deren Ergebnis wiederum der Weiterverarbeitung mittels SQL zur Verfügung steht. Der Aufruf der Suchmaschine aus einem SQL-Befehl geschieht dabei über User Defined Functions (UDFs), die Ergebnisse der Textsuche werden dem DBMS in einer virtuellen Tabelle übergeben. Durch diese Integration von Textsuchergebnis und SQL-Befehlen lassen sich Anfragen formulieren, wie z.B. “Finde alle Projekte mit Budget größer 500.000, in deren Projektbeschreibung das Wort ’Datenbank’ im gleichen Satz vorkommt wie ’objekt-relational’ ”. Diese Anfrage setzt eine Projektrelation mit Attributen für das Budget sowie für eine textuelle Projektbeschreibung voraus, wobei letztere von einer Textsuchmaschine erfasst wird, die einen “Im-gleichen-Satz”-Operator anbietet.

2.4.2 WSQ

Ein anderer Ansatz zur Verbindung von Datenbanken und Suchmaschinen wurde von Goldman und Widom [2000] unter dem Namen Web Supported Database Queries (WSQ) (etwa: Web-unterstützte Datenbankanfrage) vorgeschlagen. Er verfolgt ein ähnliches Ziel wie die Kopplung nach Dessloch und Mattos [1997]. WSQ beschränkt sich allerdings nicht auf Dokumente, die direkt der Datenbank unterstehen, sondern erschließt Suchergebnisse gewöhnlicher Websuchmaschinen für die Integration ins DBMS. Wie bei Dessloch und Mattos [1997] werden für die Kommunikation zwischen Suchmaschine und DBMS UDFs und virtuelle Tabellen verwendet.

Beiden Systemen ist gemeinsam, dass im Unterschied zu den in Kap. 2.2 vorgestellten Lösungen die Anfragen an die Suchmaschine in strukturierter Form, d.h. als Teil eines SQL-Befehls, formuliert werden müssen. Die Motivation hierfür liegt im Anwendungsbereich. Während die in Kap. 2.2 vorgestellten Lösungen dem Benutzer helfen sollen, die Struktur der Datenbank besser zu verstehen und somit als Werkzeuge im Entwicklungsprozess komplexer SQL-Anfragen Verwendung finden dürften, ermöglichen die hier vorgestellten Lösungen eine direkte Weiterverarbeitung der Ergebnisse der Suchanfragen per SQL. Ihr Anwendungsbereich liegt somit eher in Bereichen sich schnell ändernder Text-Daten, auf denen gleichbleibende SQL-Anfragen ausgeführt werden sollen.

2.4.3 SCORE

Das von Roy et al. [2005] vorgestellte SCORE-System bietet ebenfalls die Möglichkeit, per SQL auf Datenbankdaten sowie Textdokumente zuzugreifen. Im Unterschied zu den beiden vorangegangenen Ansätzen wird die Anfrage an die Textsuchmaschine nicht als Teil des SQL-Befehls ausgeführt, sondern als eigenständige Aktion. Somit werden die Ergebnisse der Suchmaschine auch nicht der Weiterverarbeitung per SQL zur Verfügung gestellt, sondern zusammen mit den Ergebnissen der Datenbankanfrage dem Benutzer angezeigt.

Das Anwendungsszenario des SCORE-Systems erstreckt sich auf Gebiete, in denen zu bestimmten Informationsbedürfnissen Daten in strukturierter sowie unstrukturierter Form vorliegen und *beide* Arten von Information gefunden werden sollen. Als Beispiel wird ein Investment-Informationssystem genannt, welches sowohl strukturierte Börsendaten in einer Datenbank als auch über ein Archiv von Nachrichtenmeldungen verfügt. Eine SQL-Anfrage an SCORE, wie bspw. “Welche Aktienkurse haben das größte Wachstum” wird zunächst an das DBMS weitergeleitet. Das Ergebnis dieser Anfrage enthält eine Menge von Tupeln, welche im Folgenden ergänzt werden durch Daten aus Tabellen, die mittels Fremdschlüsselbeziehung mit den in der Anfrage verarbeiteten Daten verbunden sind. Dieser erweiterte Datensatz wird nun auf wichtige Schlüsselworte hin analysiert, die dann an die Textsuchmaschine weitergegeben werden. Die Antwort des SCORE-Systems besteht schließlich aus den Ergebnistupeln des DBMS sowie aus den von der Textsuchmaschine gefundenen Dokumenten – für das obige Beispiel also aus einer Liste der Aktienkurse mit hohen Schwankungen sowie einer Menge von Dokumenten, die je nach ergänztem Datenbankinhalt Informationen zu den entsprechenden Unternehmen oder ihren Produkten enthalten könnten.

2.4.4 RDBMS als Suchmaschine

Der von Grossman et al. [1997] vorgestellte Ansatz ähnelt den vorangegangenen insoweit, dass auch hier Textsuchanfragen in SQL kodiert werden müssen. Während

die vorangegangenen Ansätze jedoch auf datenbankexterne Suchmaschinen zugreifen, liegt der Fokus in [Grossman et al., 1997] auf der *Implementierung* einer Textsuchmaschine mittels Datenbanktechnologie. So werden die Indexe und Hilfsstrukturen einer Textsuchmaschine in Tabellen kodiert. Schlüsselwortanfragen werden in SQL formuliert. Da das Ergebnis einer Suchanfrage von einer SQL-Anfrage berechnet wird, kann es natürlicherweise durch weitere SQL-Anweisungen mit strukturierten Daten verknüpft werden.

2.4.5 BIKM

Das von Cody et al. [2002] vorgestellte BIKM-System unterscheidet sich in mehrfacher Hinsicht von den vorangegangenen Ansätzen und wird an dieser Stelle lediglich der Vollständigkeit halber kurz erwähnt. Das Ziel des BIKM-Systems liegt darin, Business Intelligence (BI) mit Knowledge-Management (KM) -Anwendungen zu integrieren. Im konkreten Szenario steht, ähnlich zu einer von Mishne et al. [2005] vorgestellten Lösung, die Analyse von in Call-Centern erhobenen Gesprächsprotokollen im Vordergrund. Das BIKM-System erschließt Protokolle mittels Textanalysemethoden und verbindet sie mit strukturierten Unternehmensdaten. Die so integrierte Information wird in ein Data Warehouse geladen und kann somit von Data-Mining- und OLAP-Anwendungen ausgewertet werden. Mit Hilfe des BIKM-Systems können Fragen beantwortet werden, wie z.B. “Welche Gesprächsmuster versprechen eine positive Kundenbewertung?” oder “Welche Kunden erzeugen bezogen auf ihre Auftragshöhe überproportionale Call-Center-Last?”.

Während bei den oben diskutierten Lösungen die Auswertung der Daten mittels Schlüsselwort- bzw. SQL-Anfrage gesteuert wird, steht bei BIKM die Auswertung der Daten mittels OLAP-Technologie im Vordergrund. Mit BIKM können somit keine einzelnen Texte basierend auf Schlüsselwörtern gefunden werden. Vielmehr wird hier auf die statistische Auswertung von Text*merkmalen* fokussiert. Während bei den zuvor diskutierten Ansätzen die Integration ad-hoc zur Anfragezeit stattfindet, muss das Laden des Data Warehouse bei BIKM in einem separaten Schritt vor der Anfragezeit durchgeführt werden.

2.4.6 Bedeutung für den Entwurf

Die hier vorgestellten Ansätze zur Integration von strukturierten und unstrukturierten Daten verfolgen ein ähnliches Ziel wie der in dieser Arbeit vorgestellte Ansatz. Alle sind motiviert durch die derzeit unzureichende Kopplung inhaltlich zusammenhängender Daten aus beiden Bereichen.

Im Gegensatz zu dem in dieser Arbeit vorgestellten Ansatz werden jedoch unstrukturierte Daten in die Welt der strukturierten Daten eingebettet. Die Zugriffsschnittstelle besteht daher aus strukturierten SQL-Anfragen. Wie in Anforderung (1)

in Abschnitt 1.2 beschrieben, entspricht dieses jedoch nicht den Anforderungen an eine Intranetsuchmaschine.

Weiterhin steht bei der Integration von strukturierten und unstrukturierten Daten in einem Data Warehouse die Auswertung mittels OLAP-Technologie und somit eine statistische Auswertung des Dokumentenstamms im Vordergrund; es werden daher Aussagen über *Mengen* von Dokumenten getroffen. Im Gegensatz dazu ermöglicht eine Suchmaschine das Auffinden *einzelner* Dokumente.

2.5 Semantische Suche

Unter dem Begriff der semantischen oder wissensbasierten Suche versteht man im Allgemeinen eine Suche nach Dokumenten, welche von Domänenwissen Gebrauch macht. Dabei ist weder die Art oder die Struktur des Domänenwissens festgelegt noch die Art der Einflussnahme auf den Suchprozess. Ähnlich zur vorliegenden Arbeit besteht das Ziel der semantischen Suche darin, die Suche nach Dokumenten durch die Verwendung von Kontextinformation, d.h. in diesem Fall Domänenwissen, zu verbessern. Einige Konzepte aus diesem Bereich lassen sich daher auf das Szenario der vorliegenden Arbeit übertragen.

Im Bereich der semantischen Suche wurde in den vergangenen Jahren eine Vielzahl von Lösungen vorgestellt, die sich sowohl im Anwendungsgebiet als auch in ihrer Realisierung unterscheiden. Um einen Überblick über die publizierten Ansätze zu erhalten, wurde im Rahmen der vorliegenden Arbeit eine Literaturrecherche durchgeführt, in deren Rahmen 22 Ansätze analysiert wurden [Mangold, 2007]. Auf Basis dieser Recherche wurde sodann ein Kategorisierungsschema entworfen und die Ansätze dementsprechend eingeordnet.

In diesem Abschnitt wird als Grundlage zur semantischen Suche zunächst eine kurze Einführung in die Themenbereiche Semantic Web und Ontologien gegeben. Anschließend wird kurz das o.g. Schema zur Kategorisierung präsentiert und schließlich werden einige der Ansätze entlang dieser Kategorisierung vorgestellt.

2.5.1 Semantic Web und Ontologien

Die Verbreitung des World Wide Web und die damit verbundene Möglichkeit, eine unüberschaubare Menge an Information für eine große Anzahl von Benutzern zugänglich zu machen, ist heute als ein wichtiger Motor für die Entwicklung von Suchmaschinen anzusehen. In vielen Fällen jedoch erweist sich die Menge der allein durch Hyperlinks verknüpften Webseiten als zu groß und zu unübersichtlich, als dass relevante Information effizient gefunden werden könnte. Um diese Situation zu verbessern, erdachte der “Erfinder” des World Wide Web, Tim Berners-Lee, das sogenannte *Semantic Web* [Berners-Lee et al., 2001]. Die Idee des Semantic Web besteht darin, Seiten des Web mit semantischer Information anzureichern, welche so-

wohl von Menschen als auch von Maschinen "verstanden" bzw. verarbeitet werden kann.

Die dem Web hinzugefügte semantische Information wird dazu in speziellen Wissensstrukturen verwaltet, die häufig als *Ontologie* bezeichnet werden. Das Wort "Ontologie" kommt ursprünglich aus dem Griechischen und bezeichnet eine philosophische Disziplin. Im Kontext des Semantic Web wird darunter eine Wissensrepräsentation verstanden, wie sie seit langem im Bereich der Künstlichen Intelligenz (KI) verwendet wird. Die Definitionen von "Ontologie" sind vielfältig. Häufig genannt wird die Definition nach Gruber [1993]: "An ontology is an explicit specification of a conceptualization" (etwa: Eine Ontologie ist eine explizite Spezifikation einer Konzeptualisierung). Um hervorzuheben, dass eine Ontologie nur dann von Nutzen ist, wenn sie von mehreren Personen als gültig akzeptiert wird, wird oftmals von gemeinsamer Konzeptualisierung (engl.: shared conceptualization) gesprochen. Unter "Konzept" (engl.: concept, frame) wird dabei ein in einer Ontologie formalisierter Begriff verstanden. Konzepte sind untereinander durch Beziehungen oder Relationen (engl.: relation, property) verknüpft.

Aus dem Bereich der KI herkommend, wurde im Verlauf der Jahre eine Vielzahl von Ontologiebeschreibungssprachen entwickelt. Die Grundidee besteht jeweils darin, aus einer Menge formalisierten Wissens automatisch neues Wissen abzuleiten, d.h. zu inferieren oder deduzieren. Die Ontologiebeschreibungssprachen unterscheiden sich im Wesentlichen einerseits durch ihre Modellierungskonstrukte und andererseits durch die Mächtigkeit ihrer zugehörigen Inferenzmaschinen. Dabei führen umfassende Modellierungskonstrukte zu eingeschränkten Inferenzmöglichkeiten, was häufig mit "Expressiveness-Tractability-Tradeoff" bezeichnet wird [Levesque und Brachman, 1987; Baader et al., 2003].

Im Rahmen des Semantic Web wurde die Ontologiebeschreibungssprache *RDF* (für "Ressource Description Framework") [Manola und Miller, 2004] eingeführt. RDF stellt ein einfaches Graphenmodell bereit und wird durch die Schemasprache *RDFS* (für "RDF Schema") [Brickley und Guha, 2004] erweitert, die eine Festschreibung der Graphen*strukturen* ermöglicht. Die Kombination von RDF und RDFS wird oft mit RDF(S) bezeichnet.

Aufbauend auf RDF(S) wurde die Ontologiebeschreibungssprache *OWL* (für "Web Ontology Language") entwickelt [McGuiness und van Harmelen, 2004]. OWL verfügt über drei Dialekte verschiedener Modellierungsmächtigkeit, bei denen der mächtigere Dialekt jeweils den weniger mächtigen beinhaltet. Die drei Dialekte heißen in der Reihenfolge zunehmender Modellierungs- und abnehmender Inferenzmächtigkeit "OWL Light", "OWL DL" (für "Description Logics"), und "OWL Full".

Eine häufig betrachtete Anwendung im Bereich des Semantic Web besteht in der Auflösung von Homonymen, d.h. der Mehrdeutigkeit von Begriffen. Taucht auf einer herkömmlichen Webseite der Begriff "Lager" auf, so ist a priori ungeklärt ob ein Kugellager, ein Materiallager, ein Zeltlager oder vielleicht sogar keine dieser drei Bedeutungen gemeint ist. Im Semantic Web hingegen könnte für jede der Bedeutungen

ein separates Konzept existieren, sodass die Seite eindeutig zugeordnet werden kann. Bei Anfragen an eine semantische Suchmaschine spricht man in diesem Kontext von *Anfragedisambiguierung*, da die Ambiguität der Suchanfrage aufgelöst wird.

2.5.2 Merkmale semantischer Suche

In diesem Abschnitt werden einige grundlegende Merkmale semantischer Suchmaschinen eingeführt. Sie sind einer im Rahmen dieser Arbeit entstandenen Literaturrecherche [Mangold, 2007] entnommen und werden hier nur verkürzt wiedergegeben. Für eine ausführliche Diskussion zur Auswahl der Merkmale sowie deren Bedeutung sei auf die Recherche selbst verwiesen.

- Architektur. Analog zu nicht-semantischen Suchmaschinen wird zwischen zwei möglichen Architekturprinzipien unterschieden:
  - Eigener Index. Suchmaschinen mit eigenem Index realisieren den Crawler, den Indexer und den Searcher mit eigenen Mitteln, s.a. Abschnitt 2.1.
  - Meta-Suchmaschine. Eine Metasuchmaschine unterhält keinen eigenen Index. Benutzeranfragen werden lediglich modifiziert und an (potenziell mehrere) andere Suchmaschinen weitergeleitet. Die daraus resultierenden Ergebnisse werden aufbereitet und an den Benutzer zurückgegeben.

- Kopplung. Ein wichtiges Unterscheidungskriterium ist die Kopplung zwischen Dokumenten und Wissensstrukturen, d.h. Ontologien.
  - Enge Kopplung. In eng gekoppelten Systemen existiert eine explizite Verbindung (engl.: Commitment) zwischen Dokument und Wissensstruktur. Dies kann entweder durch Annotation der Dokumente mit Konzepten der Wissensstruktur oder durch Erweiterung der Wissensstruktur mit Verweisen auf Dokumente realisiert werden.
    Der Vorteil enger Kopplung liegt darin, dass das Homonymie-Problem korrekt gelöst werden kann. Der Nachteil liegt im Aufwand zur Erzeugung der Kopplungsinformation. Mit dieser Zuordnung beschäftigt sich der Forschungsbereich “Information Extraction”, siehe z.B. [Ciravegna et al., 2002; Manning und Schütze, 1999].
  - Lose Kopplung. Ein System wird als lose gekoppelt bezeichnet, falls keine explizite Verbindung zwischen Dokumenten und Wissensstruktur besteht. Verknüpfungen zwischen Text und Ontologie können nur aus der Gleichheit von Begriffen abgeleitet werden. In lose gekoppelten Systemen kann daher leicht die Ontologie ausgetauscht werden, oder es können mehrere konkurrierende Ontologien zum Einsatz kommen.

- Transparenz. Die Transparenz eines Systems wird dadurch bestimmt, inwieweit die Benutzerschnittstelle die semantischen Eigenschaften eines Systems reflektiert.

  - Transparent. Die Benutzerschnittstelle unterscheidet sich nicht von Benutzerschnittstellen nicht-semantischer Systeme.
  - Interaktiv. Der Benutzer wird direkt mit der wissensbasierten Funktionalität des Systems konfrontiert. Das System kann etwa Rückfragen an den Benutzer stellen oder Verbesserungen der Anfrage vorschlagen.
  - Hybrid. Standardanfragen werden transparent abgewickelt; für erfahrene Benutzer werden jedoch Interaktionsmöglichkeiten angeboten.

- Benutzerkontext. Die Relevanz von Dokumenten hängt stets vom Benutzerkontext ab. Bei den analysierten Systemen finden sich die folgenden beiden Ansätze, die sich gegenseitig jedoch nicht ausschließen:
  - Lernend. Der Benutzerkontext wird aus der Interaktion mit dem System abgeleitet.
  - Statisch. Anfragen werden festgelegten Kategorien zugeordnet (sog. Question Categories) und dementsprechend ausgewertet.

- Anfragemodifikation. Auf dem Gebiet der Modifikation von Anfragen existiert eine große Anzahl von Ansätzen. Wie in Abbildung 2.2 dargestellt, werden die folgenden Klassen unterschieden.
  - Manuell. Der Benutzer wird mit ausgewählten Teilen der Wissensstruktur konfrontiert und wird aufgefordert, die Suchanfrage zu verfeinern. Manuelle Anfragemodifikation ist nur in interaktiven Systemen möglich.
  - Umformulierung. Das System nimmt selbständig eine Umformulierung der Anfrage vor. Dabei kann die Anfrage sowohl erweitert als auch verkürzt werden, oder es können einzelne Suchbegriffe ersetzt werden. Bei Erweiterung bzw. Verkürzung der Anfrage ist jeweils zwischen dem konjunktiven und dem disjunktiven Fall zu unterscheiden. Unter konjunktiver Erweiterung wird das Hinzufügen von Suchbegriffen mittels UND-Verknüpfung verstanden, was i. Allg. zu einer Verkleinerung der Menge der Suchresultate und somit zu einer Präzisierung führt. Umgekehrt werden bei disjunktiver Erweiterung Suchbegriffe mittels ODER-Verknüpfung hinzugefügt. Analoges gilt für die Anfrageverkürzung.
  - Graphenbasiert. Wissensstruktur und Dokumente werden als Graphenstruktur aufgefasst. Durch die Anfrage werden einzelne Knoten im Graphen selektiert. Basierend auf der Verknüpfungsinformation des Graphen werden sodann relevante Dokumente identifiziert. Graphenbasierte Anfragemodifikation erfordert ein eng gekoppeltes System.

- Struktur der Ontologie. Semantische Suchmaschinen unterscheiden sich darin, welche ontologischen Strukturen vorausgesetzt werden. Alle untersuchten Suchmaschinen basieren ausschließlich auf Relationen.
  - Anonym. Alle Relationen werden gleichberechtigt behandelt.

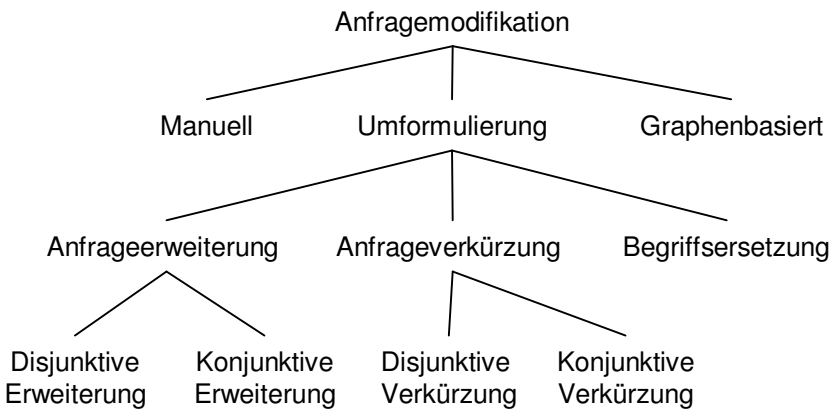


Abbildung 2.2: Varianten der Anfragemodifikation.

  - Standard. Einige wenige Relationen werden gemäß ihrer Semantik verwendet. Dazu gehören: ist-Synonym-von, ist-Hypernym-von, ist-Meronym-von, ist-Instanz-von und ist-Gegenteil-von.
  - Domänenspezifisch. Die Suchmaschine ist auf bestimmte Relationen einer ausgewählten Domäne zugeschnitten.

- Ontologiebeschreibungssprache. Während die Struktur der Ontologie die Wiederverwendbarkeit auf semantischer Ebene bestimmt, ist die Ontologiebeschreibungssprache hauptsächlich auf technologischer Ebene für die Übertragbarkeit verantwortlich. Zu den am weitesten verbreiteten Ontologiebeschreibungssprachen zählen F-Logic [Kifer et al., 1995], RDF [Manola und Miller, 2004], DAML+OIL [Horrocks, 2002] und OWL [McGuiness und van Harmelen, 2004]. Einige Ansätze verwenden zusätzlich die lexikalische Datenbank WordNet [Fellbaum, 1998].

2.5.3 Abgrenzung und ausgewählte Ansätze

Wie bereits angesprochen, existiert im Bereich der semantischen Suche eine Vielzahl von Ansätzen und Ideen. Um einen aussagekräftigen Vergleich anstellen zu können, wurde in der großen Menge der Ansätze eine Teilmenge identifiziert, die den in dieser Arbeit formulierten Anforderungen an eine Intranet-Suchmaschine am nächsten kommt.

Nicht besprochen werden bspw. die nicht-wissensbasierten Ursprünge einiger Konzepte, wie z.B. der Ansatz von Mitra et al. [1998], welcher eine automatische Anfragemodifikation vorschlägt, die auf einer statistischen Analyse des Dokumentenstamms nach häufig zusammen auftretenden Wörtern basiert, der sog. Word-Co-occurrence.

Die Ansätze der semantischen Suche wurden nicht nur auf die herkömmliche Dokumentensuche sondern auf einige weitere Bereiche übertragen. In diese Kate-

gorie fallen Ansätze zur semantischen Suche auf XML Dokumenten [Cohen et al., 2003] oder auf Peer-To-Peer-Systemen [Ding et al., 2004; Calvanese et al., 2004; Ehrig et al., 2003]. Diese werden im Folgenden ebenfalls nicht weiter betrachtet.

Weiterhin von der Betrachtung ausgenommen wird die große Klasse von Ansätzen, welche die Eingabe eines Suchausdrucks in einer formalen Sprache erfordern, wie z.B. RQL [Karvounarakis et al., 2002] oder RDQL [Seaborne, 2004] für die Ontologiebeschreibungssprache RDF. Zu ihr zählen etwa der Query-by-Example-Ansatz von Banks et al. [2002] oder der Ansatz von Calvanese et al. [2004], bei dem Benutzeranfragen in Logik erster Stufe (FOL) formuliert werden müssen.

Folgende Systeme wurden analysiert: SHOE [Heflin und Hendler, 2000], QuizRDF [Davies et al., 2004], TAP [Guha et al., 2003], Hybrid Spreading Activation [Rocha et al., 2004], ISRA [Burton-Jones et al., 2003], Librarian Agent [Stojanovic, 2003], SCORE [Sheth et al., 2002], TRUST [Amaral et al., 2004], ein System für die Recherche von Audio Daten [Khan et al., 2004], Inquirus2 [Glover et al., 2001], Ontogator [Hyvönen et al., 2003], drei Ansätze zur Anfragedisambiguierung [Mandreoli et al., 2005; Liu et al., 2005; Bai et al., 2005], WebSCSA [Crestani und Lee, 1999], OntoIR [García und Sicilia, 2003], OWLIR [Shah et al., 2002], Swangler [Finin et al., 2005], eine dynamische Inferenzmaschine [Jelmini und Marchand-Maillet, 2004], SemanticMiner [Moench et al., 2003], OntoSeek [Guarino et al., 1999], ein System zur semantischen Suche in annotierten chinesischen Gedichten [Soo et al., 2004], Freedom [Semagix, 2005] und KAON [Maedche und Motik, 2003].

Im Folgenden wird auf die für die vorliegende Arbeit relevanten Ansätze eingegangen. Auf die Bedeutung der einzelnen Ansätze für die vorliegende Arbeit wird in Abschnitt 2.5.5 weiter eingegangen. Bei der Präsentation wird aus Platzgründen darauf verzichtet, für die vorgestellten Ansätze alle im vorangegangenen Abschnitt eingeführten Merkmale einzeln zu diskutieren. Die folgende Betrachtung soll vielmehr die jeweiligen Besonderheiten hervorheben. Für die Einordnung gemäß der obigen Merkmale sei auf Tabelle 2.1 und auf die Studie selbst [Mangold, 2007] verwiesen.

2.5.3.1 SHOE und OntoIR

Ein früher Ansatz zur Dokumentensuche im Semantic Web wurde im Rahmen des SHOE Projekts vorgestellt [Heflin und Hendler, 2000]. Dabei steht SHOE für Simple HTML Ontology Extension (etwa: Einfache Erweiterung von HTML durch Ontologien). Der SHOE-Ansatz setzt eine Ontologie voraus, in der Dokumententypen durch Konzepte modelliert werden. Im Intranet eines produzierenden Unternehmens könnte die Ontologie demnach etwa die Konzepte “Arbeitsanweisung” oder “Fehlerbericht” enthalten. Die einzelnen Dokumente werden innerhalb der so aufgebauten Wissensstruktur als Instanzen der Konzepte ihrer jeweiligen Dokumententypen aufgefasst. Zusätzlich kann die Wissensstruktur beliebige Kontextinformation enthalten, wie z.B. Personennamen, Abteilungszugehörigkeiten, Kompetenzen, etc. Der SHOE Ansatz sieht vor, dass sich Webseiten mittels spezieller Annotationen in die vorge-

gebene Wissensstruktur einordnen. Diese Annotationen werden in den HTML-Code der jeweiligen Webseite eingebettet und sind in der normalen Browser-Ansicht für Benutzer unsichtbar.

Die semantische Suche nach SHOE entspricht einer Navigation in der Wissensstruktur. Der Benutzer wählt ein Konzept aus, woraufhin das System die Menge der für dieses Konzept vorgesehenen Beschreibungsfelder anzeigt, die sodann vom Benutzer ausgefüllt werden. Wählt ein Benutzer z.B. das Konzept "Fehlerbericht" und weist dem Feld "Maschine-ID" den Wert "M18" zu, so wird vom System eine Liste aller Fehlerberichte der Maschine "M18" angezeigt.

Das OntoIR System [García und Sicilia, 2003] ähnelt in seiner Grundidee stark dem SHOE System. Bei OntoIR liegt der Fokus jedoch auf der Benutzerschnittstelle des Systems. Im Gegensatz zu SHOE, welches sich einer proprietären Ontologiebeschreibungssprache bedient, arbeitet OntoIR mit beliebigen RDF und DAML+OIL Ontologien.

Weder SHOE noch OntoIR verfügen über Mechanismen zur Bewertung von Ergebnisdokumenten. Bei beiden wird der Benutzer unmittelbar mit den Strukturen der zugrundeliegenden Ontologie konfrontiert und kann auf deren Basis Filter auf den Dokumentenstamm anwenden.

2.5.3.2 QuizRDF

Das QuizRDF-System von Davies et al. [2004] nutzt ähnlich wie SHOE und OntoIR die Annotation von Webseiten mit RDF-Ressourcen aus. Weiterhin ähnlich zum SHOE-Ansatz bietet QuizRDF einen interaktiven Suchprozess an, bei dem der Benutzer, vom System unterstützt, durch die Wissensstruktur navigiert. Im Unterschied zu SHOE basiert QuizDRF auf einer gewöhnlichen Suchmaschine, deren Index um die zusätzlich verfügbare Information der Wissensstruktur erweitert wird. Auf diese Weise ist QuizRDF in der Lage, annotierte sowie nicht-annotierte Dokumente im selben Index abzulegen. Somit kann sich das Ergebnis einer Suchanfrage aus annotierten und nicht-annotierten Dokumenten zusammensetzen.

2.5.3.3 TAP

Das Semantic-Web-Framework TAP kombiniert herkömmliches Information Retrieval mit semantischer Suche [Guha et al., 2003]. Das Semantic Web wird bei TAP als eine RDF-Ontologie betrachtet, die nicht direkt mit den Webseiten verknüpft ist. Die semantische Suche ist ein unabhängiger Zusatz zur normalen Suche; d.h. jedes Suchergebnis besteht aus zwei Teilen: Zum einen wird eine Liste von Dokumenten angezeigt, welche mit Mitteln herkömmlicher Suche gefunden wurde. Zum anderen wird der Teil der Ontologie angezeigt, welcher die gesuchten Begriffe beschreibt. Der Swoogle Ansatz von Ding et al. [2004] konzentriert sich ebenfalls auf die Suche von Konzepten in Ontologien; da jedoch nur Konzepte und keine Dokumente

recherchiert werden, wird an dieser Stelle auf eine weitere Diskussion verzichtet.

Das Problem der Anfrage-Disambiguierung wird bei TAP mit dreierlei Mitteln angegangen. Zum einen werden die Abstände zwischen Suchbegriffen basierend auf dem kürzesten sie verbindenden Weg im RDF-Graph gemessen. Diese Technik beruht auf der Idee, dass Homonyme, also die Bezeichnung zweier Konzepte mit dem gleichen Begriff, dadurch aufgelöst werden können, dass sich Begriffe einer Suchanfrage gegenseitig kontextualisieren und somit oftmals in einer Wissensstruktur verwandte Konzepte bezeichnen. Taucht bspw. das Homonym "Lager" (für Kugellager bzw. Materiallager) in der Anfrage "Lager Vorrat" auf, so kann vom System das wahrscheinlichere Konzept des Materiallagers ausgewählt werden. Zum zweiten wird der Benutzerkontext hinzugezogen. Dies wird einerseits über die Verwendung der Anfragehistorie realisiert, andererseits kann TAP Teile der Ontologie für bestimmte Benutzer(-Gruppen) als relevant markieren. Zum dritten bietet TAP schließlich die Möglichkeit, die Popularität von Konzepten der Ontologie, d.h. deren Verwendungshäufigkeit, in die Disambiguierung einfließen zu lassen.

2.5.3.4 Hybrid Spreading-Activation

Ähnlich wie bei SHOE und OntoIR werden beim Hybrid-Spreading-Activation-Ansatz von Rocha et al. [2004] Webseiten als Individuen einer Ontologie betrachtet. Die Ontologie wird in diesem Ansatz auf ihre Darstellung als Graph reduziert. Die Knoten im Graph entsprechen den Individuen der Ontologie; die Kanten ihren Beziehungen. Zusätzlich werden die Kanten mit einem Maß für den semantischen Abstand zwischen den beiden durch sie verbundenen Knoten gewichtet. Suchanfragen werden in zwei Schritten ausgeführt. Im ersten Schritt werden mittels einer Textsuchmaschine die Knoten des Graphen bestimmt, welche einen der Suchbegriffe enthalten. Diese Knotenmenge wird im zweiten Schritt als Ausgangsmenge für einen Spreading-Activation Algorithmus (siehe Abschnitt 2.7.1) eingesetzt. Der Aktivierungsgrad von Knoten im Graph spiegelt dann deren Relevanz bzgl. der Anfrage wider. Dementsprechend wird die Relevanz eines Dokuments durch den Aktivierungsgrad des entsprechenden Knoten bestimmt.

Die Ausführung des Spreading-Activation-Algorithmus wird von Rocha et al. [2004] jedoch zumindest als performance-kritisch eingestuft: "We are aware that spread activation algorithms are highly costly [...]" (etwa: Wir wissen, dass Spreading-Activation-Algorithmen höchst aufwändig sind [...]). Es konnten keine Laufzeitmessungen zur Evaluierung des Systems gefunden werden.

Crestani und Lee [1999] schlagen den sehr ähnlichen WebSCSA-Ansatz vor, der jedoch keine Schlüsselwortsuche, sondern die Suche nach ähnlichen Dokumenten realisiert. Da die wesentlichen Ideen beider Ansätzen übereinstimmen, wird an dieser Stelle auf eine weitere Diskussion verzichtet.

2.5.3.5 ISRA

Der Intelligent-Semantic-Web-Retrieval-Agent (ISRA) ist als semantische Meta-Suchmaschine realisiert [Burton-Jones et al., 2003]. Jede Suchanfrage wird von ISRA in ein kleines semantisches Netz transformiert, welches mit im System vorgehaltenen DAML-Konzepthierarchien und mit Informationen aus der WordNet-Datenbank abgeglichen wird. Auf diesem Abgleich basierend, werden in der Anfrage etwaig auftretende Inkonsistenzen und Homonyme erkannt und aufgelöst. Eine modifizierte Anfrage wird sodann an die der Meta-Suchmaschine untergeordneten Systeme weitergeleitet.

2.5.3.6 Librarian Agent

Das Librarian Agent System von Stojanovic [2003] imitiert das Verhalten eines menschlichen Bibliothekars und hilft so dem Benutzer in einem interaktiven Prozess, die gewünschten Ressourcen zu lokalisieren. Die Anfrageverarbeitung greift dabei auf drei Arten von Informationen zurück: Zum einen wird eine Ontologie eingesetzt, um die Anfrage zu disambiguieren und ggf. beim Benutzer nachzufragen, welche Wortbedeutung gemeint ist. Zum zweiten wird die Anfragehistorie ausgenutzt, um das bevorzugte Themengebiet des Benutzers zu bestimmen. Zum dritten wird schließlich der Dokumentenstamm selbst hinzugezogen, um den Benutzer bei der Verbesserung seiner Anfrage zu unterstützen. Sucht ein Benutzer bspw. nach Klebenaht UND Hubhebel, so zeigt das System neben den 25 Resultatdokumenten etwa den Hinweis an, dass die verkürzte Suche nur nach Klebenaht bzw. Hubhebel, 156 bzw. 289 Treffer erzielt, die erweiterte Suche nach Klebenaht UND Hubhebel UND links jedoch zehn relevante Dokumente findet. Wie ein Bibliothekar, der den Inhalt seiner Bibliothek genau kennt, unterstützt das System auf diese Weise den Benutzer bei der Lokalisierung der gewünschten Information.

2.5.3.7 SCORE

Die Semantic-Content-Organization-and-Retrieval-Engine SCORE [Sheth et al., 2002] hat einen breiteren Fokus als viele der anderen betrachteten Ansätze. Während oftmals die Existenz einer domänenspezifischen Wissensstruktur sowie die Zuordnung von Dokumenten zu Elementen derselben als gegeben vorausgesetzt wird, steht bei SCORE u.a. die Erzeugung dieser Information im Vordergrund. Für die eigentliche semantische Suche muss bei SCORE jedoch dem Benutzer die Struktur der Ontologie bekannt sein. Der SCORE Ansatz ist einer der wenigen, von denen bekannt ist, dass sie zur Produktreife gelangten. So bot die Firma Semagix (heute: Fortent) für einige Zeit das auf SCORE basierende Produkt FREEDOM an [Semagix, 2005].

Prototyp / Projekt	**SHOE** [Heflin und Hendler, 2000]	**QuizRDF** [Davies und Weeks, 2004]	**TAP** [Guha et al., 2003]	**Hybrid Spreading Activation** [Rocha et al., 2004]	**ISRA** [Burton-Jones et al., 2003]	**Librarian Agent** [Stojanovic, 2003]	**SCORE** [Sheth et al., 2002]	**TRUST** [Amaral et al., 2004]
Fokus	WWW	WWW	WWW	WWW	WWW	WWW	IS	IS
Architektur	eigener Index	eigener Index	Meta	eigener Index	Meta	eigener Index	eigener Index	hybrid
Kopplung	eng	hybrid	lose	eng	lose	eng	eng	hybrid
Transparenz	interaktiv	interaktiv	hybrid	transparent	hybrid	interaktiv	interaktiv	unklar
Benutzer-kontext	–	–	statisch und lernend	–	–	lernend	unklar	statisch
Anfrage-modifikation	manuell	manuell	- Ontologie: graphen-basiert - Dokumente: unklar	graphen-basiert	Alle Arten von Umfor-mulierung	- konj. Verk. - konj. Erw. - Ersetzung	manuell	konj. Erw.
Struktur der Ontologie	- Hypernym - anonym	- Hypernym - anonym	anonym	domänen-spez.	- Hypernym - Synonym - Gegenteil	anonym	- Hypernym - domänen-spez.	- Hypernym - Synonym - Rest unklar
Ontologie-beschreibungs-sprache	proprietär	RDF(S)	RDF	unklar	DAML + Word Net	proprietär	unklar	proprietär

Tabelle 2.1: Ansätze zur semantischen Suche im Vergleich.

2.5.3.8 TRUST

Im Rahmen des TRUST (Text Retrieval Using Semantic Technologies) -Projekts entstand eine mutlilinguale semantische Suchmaschine. Die TRUST-Suchmaschine ist in der Lage, sowohl annotierte als auch unannotierte Dokumente zu durchsuchen. Die Anfragemodifikation basiert auf Konzepthierarchien, linguistischer Information, wie z.B. Synonymen und vordefinierten Anfragekategorien.

2.5.4 Kategorisierung und Fazit

Anhand der in Abschnitt 2.5.2 vorgestellten Merkmale von semantischen Suchmaschinen werden in diesem Abschnitt die in 2.5.3 besprochenen Ansätze kategorisiert, siehe Tabelle 2.1. Für eine detailliertere Diskussion der Einordnung und die Einordnung weiterer Ansätze sei auf die Literaturrecherche [Mangold, 2007] verwiesen.

Falls ein System wegen fehlender Information nicht eindeutig kategorisiert werden konnte, wird dies in der Tabelle mit “unklar” vermerkt. Systeme, die in einer Kategorie mehreren Klassen zuzuordnen sind, werden mit “hybrid” bezeichnet.

Die erste Zeile der Tabelle beinhaltet den Anwendungsfokus des Systems, der nicht in erster Linie ein Unterscheidungsmerkmal darstellt, sich jedoch für einen besseren Überblick als hilfreich erwiesen hat. Dabei steht “WWW” für Systeme, die für das (Semantic) Web entwickelt wurden, “IS” bezeichnet Suchmaschinen auf Informationssystemen mit abgeschlossenem Dokumentenstamm.

Bezüglich Architektur und Kopplung kann festgehalten werden, dass die Kombinationen von enger Kopplung mit eigenem Index und von Metasuchmaschinen mit loser Kopplung vorherrschen. Es wurde kein Ansatz gefunden, welcher eine eng gekoppelte Metasuchmaschine realisiert; d.h. keine der analysierten Metasuchmaschinen ist in der Lage, andere semantische Suchmaschinen auf semantischer Ebene zu integrieren.

Als Fazit der Literaturrecherche wurden unter anderem Forschungs- und Weiterentwicklungsbedarf in den folgenden Bereichen festgestellt:

- Adaptierbarkeit. Viele der untersuchten Systeme setzen bestimmte Ontologiestrukturen voraus, d.h. sie basieren auf maßgeschneiderten und teilweise domänenspezifischen Ontologien. Die mit "anonym" klassifizierten Systeme hingegen arbeiten mit Ontologien beliebiger Struktur, nutzen jedoch die darin enthaltene Semantik nicht ausschöpfend. Es konnte kein Ansatz gefunden werden, welcher sich dynamisch an existierende Ontologiestrukturen anpassen lässt. Diese Adaptierbarkeit ist insbesondere im Hinblick auf die Wiederverwendung von Ontologien, welche für andere Verwendungszwecke entworfen wurden, von Interesse.

- Integration mit DMS/CMS. Alle analysierten Systeme konzentrieren sich entweder auf den Einsatz im (Semantic) Web oder auf die Verwendung in proprietären Informationssystemen. Es konnte kein Ansatz gefunden werden, der sich mit am Markt verfügbaren Dokumentenmanagementsystemen (DMS) bzw. Contentmanagementsystemen (CMS), wie z.B. IBM ContentManager [Choy, 2005], integrieren lässt.

- Ranking. Überraschenderweise existieren kaum Ideen zur Relevanzbewertung von Suchergebnissen. Es existieren zwar einzelne Ansätze [Rocha et al., 2004; Khan et al., 2004], verglichen mit herkömmlichem Information Retrieval, wo das Ranking von Suchergebnissen zu den am meisten diskutierten Problemen gehört, wird diese Fragestellung kaum thematisiert.

- Performanz/Skalierbarkeit. Nur wenige der analysierten Ansätze wurden bzgl. Performanz und Skalierbarkeit oder Qualität der Suchresultate evaluiert. Dies ist umso erstaunlicher, als semantische Suchmaschinen als Konkurrenz zu herkömmlichen Suchmaschinen gesehen werden müssen. Einem erhöhten Ressourcenverbrauch einer semantischen Suchmaschine muss daher stets eine Verbesserung der Suchergebnisse gegenüberstehen.

2.5.5 Bedeutung für den Entwurf

Die in diesem Abschnitt diskutierten Ansätze zur semantischen Suche sowie der daraus abgeleitete Weiterentwicklungsbedarf haben vielerlei Auswirkungen auf den Entwurf des in dieser Arbeit vorgestellten Ansatzes.

Von den Systemen SHOE, OntoIR, QuizRDF u.a. wird der Ansatz übernommen, dass Dokumente sowie Kontextinformation als Knoten desselben Graphen modelliert werden. Im Unterschied zu den genannten Ansätzen wird in der vorliegenden Arbeit jedoch vom Benutzer keine direkte Interaktion mit der Wissensstruktur verlangt. Weiterhin ist QuizRDF von Interesse, da dort Informationen aus Dokumenten und damit zusammenhängende Kontextinformation in einem Index einer nicht-semantischen Textsuchmaschine abgelegt werden.

Der Ansatz des Hybrid-Spreading-Activation verbindet herkömmliche Dokumentensuche mit der Suche in Wissensstrukturen. Zu diesem Zweck wird ein Maß für den semantischen Abstand zweier Knoten in einer als Graph dargestellten Ontologie entwickelt. Dieser Ansatz ist ebenfalls für die vorliegende Arbeit anwendbar. Von der Realisierung der Suche mittels eines Spreading-Activation-Algorithmus, welcher zur Anfragezeit durch die Wissensstruktur navigiert, wurde jedoch aus Performanzgründen abgesehen. Auf Spreading-Activation als Mittel zur graphbasierten Relevanzbewertung wird in Abschnitt 2.7.1 weiter eingegangen.

Die Disambiguierung im TAP-System weist Parallelen zu der in dieser Arbeit inhärent verwendeten Disambiguierung auf. Beide basieren auf der Annahme, dass sich Begriffe einer Suchanfrage gegenseitig kontextualisieren, eine Diskussion dazu findet sich in Abschnitt 3.2.1.3. Die Ansätze ISRA und Librarian Agent finden keine direkte Umsetzung im in dieser Arbeit vorgestellten System. Sie wurden exemplarisch ausgewählt, um mögliche modulare Erweiterungen des Systems aufzuzeigen. Beide Systeme beruhen im Wesentlichen auf Funktionalität, die zur Anfragezeit zum Einsatz kommt und von Indexstrukturen unabhängig ist. In Abschnitt 4.1 wird diskutiert, an welcher Stelle das im Rahmen der vorliegenden Arbeit erstellte System um derlei Funktionalität erweitert werden kann. Analog dazu kann das System zur Indexierungszeit um Funktionalität ergänzt werden, wie sie etwa beim SCORE System zum Aufbau von Wissensstrukturen realisiert wird. Das TRUST System wurde in die Diskussion mit aufgenommen, da es (neben QuizRDF) als eines der wenigen Systeme in der Lage ist, annotierte und nicht-annotierte Dokumente in einer Suchmaschine zu integrieren und in seiner Gesamtkonzeption der in dieser Arbeit vorgestellten modularen Systemarchitektur am nächsten kommt, siehe Abschnitt 4.1.

2.6 Wissen im Unternehmen

Im Rahmen des Sonderforschungsbereichs 467[6] zum Thema “Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion” wurden theoretische Grundlagen im Bereich der unternehmensinternen Handhabung von Wissen erarbeitet. In diesem Abschnitt werden die Ergebnisse dieser Arbeiten skizziert und sodann an ihnen die Funktion einer Intranet-Suchmaschine als Werkzeug des betrieblichen Wissensmanagements erläutert.

[6] http://www.sfb467.uni-stuttgart.de/ [30.01.2006]

2.6.1 Das Stuttgarter Wissensmodell

Der Sonderforschungsbereich 467 hatte u.a. zum Ziel, ein Unternehmensmodell für produzierende Unternehmen im turbulenten Umfeld zu erstellen. Im Fokus der Untersuchungen standen somit Strukturen und Prozesse, die ein Unternehmen unterstützen, flexibel auf innere wie äußere Turbulenzen reagieren zu können. In die Kategorie der inneren Turbulenzen fallen dabei Anforderungen, die aus Personenausfall, Maschinenausfall oder unternehmensinternen Restrukturierungen sowie Strategiewechseln resultieren. Zu den äußeren Turbulenzen zählen hingegen sich ändernde Kundenwünsche, Technologiewechsel oder Gesetzesänderungen, welche eine Änderung der Produktion erforderlich machen. Grundsätzlich sind dabei jeweils kurz-, mittel- und langfristige Szenarien denkbar.

Als ein wichtiges Werkzeug zur Unterstützung dynamischer Unternehmensstrukturen wird allgemein die Verwaltung des unternehmensinternen Wissens verstanden, z.B. [Schroeder et al., 2002]. Im Teilprojekt A6 zum Thema "Wissensbasiertes Management für die wandlungsfähige Montage" des Sonderforschungsbereichs wurde das sogenannte "Stuttgarter Wissensmodell" entworfen, welches unternehmensinternes Wissen insbesondere im Hinblick auf turbulente Unternehmensumfelder strukturiert und somit als Grundlage für das Wissensmanagement verwendet werden kann [Berger et al., 2005b, 2004]. Die dem Modell zugrundeliegenden Unterscheidungskriterien sind keineswegs neu, sondern wurden aus der großen Menge von in der Literatur vorgeschlagenen (Partial-)Modellen ausgewählt und auf die im Sonderforschungsbereich behandelten Szenarien zugeschnitten.

Abbildung 2.3 zeigt einen zweidimensionalen Ausschnitt aus dem eigentlich dreidimensionalen Stuttgarter Wissensmodell. Die hier dargestellten Dimensionen zeigen den Wissensexplikationsgrad sowie die Wissenseigenschaften an. Da die dritte Dimension des Stuttgarter Wissensmodells – die Wissensobjekte – für die vorliegende Arbeit nicht von Interesse ist, wird auf eine Darstellung verzichtet.

In der Dimension der *Wissenseigenschaften* wird zwischen Kernwissen und Integrationswissen unterschieden [Helfat und Raubitschek, 2000; Henderson und Clark, 1990]. Das Kern- (auch Komponenten-) wissen wird dabei als das Verständnis elementarer Technologien und Prozesse definiert. Integrationswissen umfasst hingegen die Fähigkeit zur Rekombination bzw. zur Übertragung von Kernwissen auf andere Bereiche. Integrationswissen charakterisiert sich somit als Fähigkeit, in turbulenten Situationen durch die Identifikation von Anknüpfungspunkten und durch die Anwendung von Analogien zu bestehendem Kernwissen neue Lösungswege zu finden. Durch die Konsolidierung und Formalisierung von Integrationswissen kann so wiederum Kernwissen auf einem neuen Themengebiet erzeugt werden. Eine empirische Umfrage im Rahmen einer Dissertation [Meyer, 2007] ergab, dass insbesondere seitens Unternehmen im turbulenten Umfeld die Erzeugung von Integrationswissen als wesentlicher Erfolgsfaktor eingestuft wird.

Der *Wissensexplikationsgrad* gibt an, in welcher Weise das Wissen im Unter-

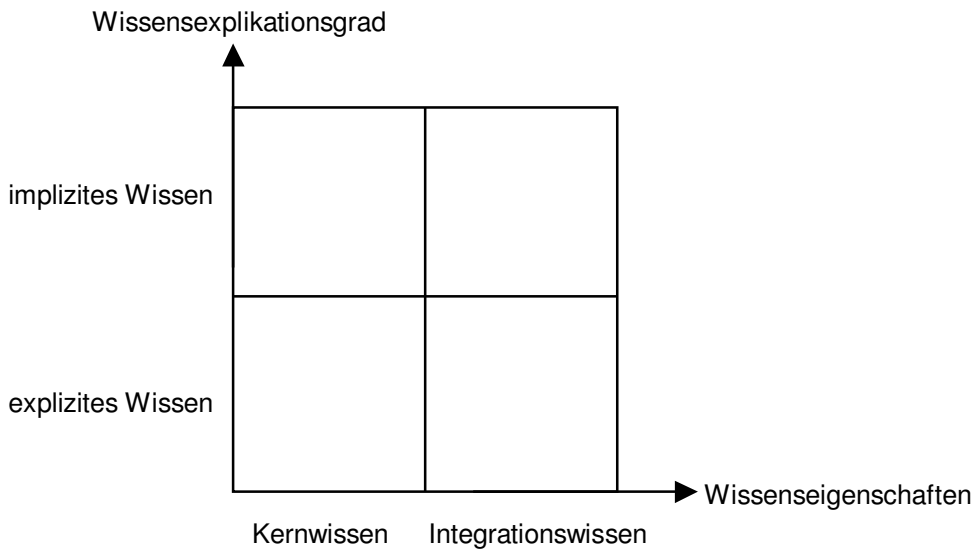


Abbildung 2.3: Ausschnitt aus dem Stuttgarter Wissensmodell nach [Berger et al., 2004].

nehmen verfügbar ist [Polanyi, 1985]. Explizites Wissen lässt sich mit vertretbarem Aufwand in Dokumenten und Datenbanken kodifizieren und somit dokumentieren und abspeichern. Explizites Wissen schlägt sich somit bspw. in Form von Berichten, Kenngrößen oder Arbeitsanweisungen nieder. Implizites Wissen hingegen bindet sich stets an eine Person oder eine Personengruppe. Es lässt sich kaum strukturiert erfassen und befindet sich in erster Linie in den Köpfen der Mitarbeiter. Implizites Wissen umfasst daher den Erfahrungsschatz des Unternehmens und insbesondere informelles Expertenwissen. Für die Wandlungsfähigkeit von Unternehmen ist von Interesse, dass beide Arten von Wissen im Unternehmen gleichermaßen auffindbar sind. Es müssen also einerseits relevante Informationsquellen zur Bereitstellung expliziten Wissens, wie auch themenbezogene Experten als Träger impliziten Wissens identifiziert und verfügbar gemacht werden.

2.6.2 Bedeutung für den Entwurf

Der in dieser Arbeit vorgestellte Ansatz zur kontextbasierten Dokumentensuche im Intranet basiert direkt auf der Wissensstrukturierung des Stuttgarter Wissensmodells und hat die Unterstützung des betrieblichen Wissensmanagements zum Ziel, vgl. Berger et al. [2005a]. Die Wirkungsweise des hier vorgestellten Ansatzes lässt sich anhand der Wissenseigenschaften und des Wissensexplikationsgrads beschreiben.

Entlang der Dimension der *Wissenseigenschaften*, welche zwischen Kern- und Integrationswissen unterscheidet, können textbasierte Dokumentensuchmaschinen zu gegebenen Suchbegriffen lediglich Dokumente finden, welche die gegebenen Suchbegriffe enthalten. Diese herkömmliche Art der Suche bildet somit die Basis für das Wiederauffinden von Kernwissen. Vor allem im turbulenten Umfeld birgt dies das Risiko, dass keine relevanten Dokumente gefunden werden können, da dokumen-

tiertes Kernwissen lediglich zu in der Vergangenheit bereits aufgetretenen Situationen hinterlegt ist. Dokumente mit ähnlichem Hintergrund bzw. Kontext werden somit missachtet und stehen dem betrieblichen Wissensmanagement nicht zur Verfügung. Eine kontextbasierte Suchmaschine hingegen, kann und soll in diesem Fall Dokumente finden, welche nur indirekt, d.h. per Kontextinformation, mit den gegebenen Suchbegriffen zusammenhängen. Falls also zu einer Fragestellung kein direkt relevantes Dokument existiert, kann eine Anzahl verwandter Dokumente angezeigt werden, die dem Kontext der Fragestellung entnommen sind. Der Anwender erhält somit Anknüpfungspunkte an expliziertes Kernwissen und wird auf diese Weise in der Bildung neuen Integrationswissens unterstützt.

Bezüglich des *Wissensexplikationsgrads* kann das betriebliche Wissensmanagement ebenfalls von den Eigenschaften einer kontextbasierten Suchmaschine profitieren. Während herkömmliche Suchmaschinen lediglich isolierte Dokumente finden können, ergibt sich durch die Betrachtung des Kontexts im Intranet ein Vorteil bei der Lokalisierung sowohl expliziten als auch impliziten Wissens.

Das Auffinden expliziten Wissens profitiert dabei von der Vernetzung strukturierter Daten mit unstrukturierten Daten nach dem Beispiel in Abschnitt 1.1.

Das Auffinden impliziten Wissens konzentriert sich auf die Identifikation von Expertenwissen einzelner Mitarbeiter. Im Bereich der Informatik beschäftigt sich der Bereich der Expertensuche mit dieser Problematik. In diesem Themengebiet existiert ein breites Spektrum an Ansätzen, welches in dieser Arbeit nicht behandelt werden kann, siehe z.B. [Yimam-Seid und Kobsa, 2003; Heeren und Sihn, 2002].

Es sei jedoch darauf hingewiesen, dass in direkter Analogie zur Kontextualisierung von Dokumenten durch Information aus Datenbanken auf der selben Basis die Kontextualisierung von Experten durch ihnen zugeordnete Dokumente realisiert werden kann. Diese Kontextualisierung könnte dann in die Suche nach Experten und somit in die Recherche impliziten Wissens miteinfließen.

2.7 Graphenbasierte Abstandsmaße

In diesem Abschnitt werden zwei konkurrierende Ansätze zur Abstandsbestimmung in Graphen vorgestellt: Spreading Activation und die Berechnung der kürzesten Wege. Beide werden in der Literatur verwendet, um die gegenseitige Relevanz von Knoten eines Graphen zu berechnen. Für die vorliegende Arbeit sind sie als Alternativen zur Auswertung von im ContextGraph abgelegter Information zu betrachten.

2.7.1 Spreading Activation

Ursprünglich aus dem Bereich der Psychologie [Collins und Loftus, 1975] entstammt der Ansatz der *Spreading Activation* (etwa: sich ausbreitende Aktivierung), der durch neurologische Modelle inspiriert wurde und der über den Bereich der Künstlichen In-

telligenz ebenfalls im Information Retrieval betrachtet wurde und wird, siehe bspw. [Stillings et al., 1987; Salton und Buckley, 1988; Crestani, 1997; Crestani und Lee, 1999; Rocha et al., 2004].

Es existiert eine Vielzahl von Spreading-Activation-Algorithmen, die jedoch alle auf folgender Idee basieren: Gegeben sei ein Graph $G = (V, E)$ mit Knoten V und Kanten E. Jedem Knoten $v_i \in V = \{v_1, \ldots, v_n\}$ wird ein Aktivierungswert a_i^0 zugeordnet, der zur Initialisierungszeit festgelegt wird. Dieser Aktivierungswert wird während der Abarbeitung des Algorithmus iterativ entsprechend der Aktivierungswerte seiner im Graphen benachbarten Knoten angepasst. In der Regel werden zur Initialisierungszeit lediglich einige wenige Knoten aktiviert, d.h. mit Aktivierungswerten größer 0 versehen. Während der Abarbeitung berechnet sich der Aktivierungswert a_i^{n+1} eines Knoten v_i in der $(n+1)$-ten Iteration wie folgt:

$$a_i^{n+1} = \sum_{v_j \in V_i} f(a_j^n \cdot \mathrm{dist}(\langle v_j, v_i \rangle))$$

Dabei ist $V_i \subseteq V$ die Menge der Knoten, die mit v_i durch eine Kante $\langle v_j, v_i \rangle$ verbunden sind, a_j^n der Aktivierungswert von a_j in der n-ten Iteration und $\mathrm{dist}(\langle v_j, v_i \rangle)$ das Gewicht der Kante $\langle v_j, v_i \rangle$. Über die Funktion f kann eine beliebige Anpassung der Aktivierungswerte vorgenommen werden.

Falls die Anzahl der Iterationen beschränkt ist, spricht man von einem Constraint-Spreading-Activation-Algorithmus. Die Funktion f ist dafür oft etwa folgendermaßen definiert:

$$f(x) = \begin{cases} 0,9 \cdot x & \text{, falls } x > 0,2 \\ 0 & \text{, sonst} \end{cases}$$

Ein Knoten gibt somit keine Aktivierung weiter, wenn seine eigene Aktivierung unter dem Schwellwert von 0.2 liegt. Der Dämpfungsfaktor $0,9$ führt dazu, dass die Aktivierung ständig abnimmt. Der Algorithmus terminiert, sobald die Aktivierung aller Knoten unter dem Schwellwert liegt und somit keine Aktivierung weitergegeben wird.

Die Laufzeit des Spreading-Activation-Algorithmus hängt somit von der Anzahl der Knoten und Kanten des Graphen sowie von der Anzahl der Iterationen s ab und liegt in $O(s \cdot (|V| + |E|))$, da in jeder Iteration potenziell sämtliche Knoten und Kanten betrachtet werden müssen.

Den Ansätzen, die Spreading-Activation-Algorithmen zur Dokumentensuche verwenden, liegt zumeist ein semantisches Netz zugrunde, welches Dokumente sowie Schlüsselworte enthält. Der initiale Aktivierungswert der Knoten wird dabei durch eine Volltextsuche auf den textuellen Inhalten der Knoten bestimmt. So werden für eine gegebene Suchanfrage alle Knoten aktiviert, die mindestens ein Schlüsselwort der Anfrage enthalten. Nach Ablauf des Algorithmus werden dann die Dokumente entsprechend ihres Aktivierungswertes bewertet und sortiert.

Dieser Ansatz wird, wie in Abschnitt 2.5.3.4 zur semantischen Suche mit Spreading Activation diskutiert, von Rocha et al. [2004] verfolgt und dürfte für größere Datenmengen kaum geeignet sein, da zur Anfragezeit eine aufwändige Navigation der Graphenstruktur erforderlich ist.

Eine Vorausberechnung der Aktivierungswerte zur Indexierungszeit stellt aus folgendem Grund keine echte Alternative dar: Diese Vorausberechnung müsste alle Kombinationen aller Suchbegriffe erfassen, was exponenziellen Aufwand sowie exponenziellen Speicherplatz in der Anzahl der möglichen Suchbegriffe impliziert.

Die Idee des Spreading-Activation-Verfahrens findet sich ebenfalls in der Berechnung des PageRanks wieder, der die Wichtigkeit von Webseiten basierend auf der Hyperlink-Verknüpfungsstruktur des Web bewertet. Auf die Berechnung des PageRank wird in Abschnitt 3.2.2.1 weiter eingegangen. Der Unterschied des hier besprochenen Einsatzes von Spreading-Activation-Algorithmen im Information Retrieval gegenüber der Berechnung des PageRanks liegt darin, dass der Aktivierungswert bei den hier angesprochenen Verfahren von den Suchbegriffen der Anfrage abhängt, während der PageRank ein von der Anfrage unabhängiges Wichtigkeitsmaß berechnet.

2.7.2 Kürzeste Wege

Die Berechnung der kürzesten Wege in einem Graphen gehört zu den klassischen Problemen der Informatik. Zu unterscheiden sind Single-Source-Shortest-Path (SSSP) Algorithmen, die die kürzesten Wege von einem vorgegebenen Startpunkt aus berechnen und All-Pairs-Shortest-Path (APSP) Algorithmen, die die kürzesten Wege zwischen allen Knotenpaaren eines Graphen berechnen.

In diesem Abschnitt wird zunächst kurz der Algorithmus nach Dijkstra als fundamentaler SSSP Algorithmus diskutiert und anschließend der HiddenPath-Algorithmus zur Berechnung von APSP. Im Protoyp wurde neben diesen beiden Algorithmen ebenfalls der Algorithmus nach Floyd-Warshall implementiert, siehe z.B [Cormen et al., 1994, Kap. 26.2]. Da dieser jedoch keinerlei Optimierungspotenzial bietet und bzgl. Performanz den beiden hier vorgestellten Algorithmen klar unterliegt, wird auf eine Vorstellung verzichtet.

2.7.2.1 Algorithmus nach Dijkstra

Der Single-Source-Shortest-Path Algorithmus nach Dijkstra ist ein viel diskutierter und oft verwendeter Standardalgorithmus zur Kürzesten-Wege-Suche auf gerichteten, gewichteten Graphen mit positiven Kantengewichten. Auf eine Vorstellung soll an dieser Stelle verzichtet werden; für eine Einführung sei auf eines der zahlreichen Lehrbücher, z.B. [Cormen et al., 1994, S. 527], verwiesen.

Eine wesentliche und für die Zwecke der vorliegenden Arbeit notwendige Eigenschaft des Algorithmus nach Dijkstra ist, dass die kürzesten Wege in der Reihenfolge

aufsteigender Gewichte berechnet werden. Durch diese Eigenschaft ist es möglich, lediglich die kürzesten Wege innerhalb einer beschränkten Umgebung des Startknoten zu berechnen.

Wie alle Single-Source-Shortest-Path Algorithmen kann auch der Algorithmus nach Dijkstra leicht zum All-Pairs-Shortest-Path Algorithmus ausgebaut werden, indem er sequenziell für alle Knoten des Graphen gestartet wird. Für den Rest der Arbeit wird dieser iterierte Algorithmus mit "n-facher Dijkstra" bezeichnet.

Die Komplexität des Algorithmus nach Dijkstra hängt wesentlich von der Umsetzung der im Algorithmus verwendeten Prioritätswarteschlange ab. Wird diese mittels eines Fibonacci-Heaps [Cormen et al., 1994, Kap.21] realisiert, liegt die Laufzeit auf einem Graphen mit n Knoten und m Kanten in $O(m + n \lg n)$, die Laufzeit des n-fachen Dijkstra somit in $O(nm + n^2 \lg n)$.

2.7.2.2 HiddenPath Algorithmus

Der HiddenPath-Algorithmus wurde von Karger et al. [1993] vorgeschlagen. Er basiert auf der Erkenntnis, dass beim n-fachen Dijkstra Information teilweise mehrfach berechnet wird. Durch Wiederverwendung dieser Information erzielt der HiddenPath-Algorithmus gegenüber dem n-fachen Dijkstra eine Laufzeitreduktion.

Aus logischer Sicht wird für jeden Knoten des Graphen ein eigener Dijkstra-Algorithmus gestartet. Dies geschieht jedoch nicht streng sequenziell, sondern ineinander verschränkt. Diese gegenseitige Verschränkung ermöglicht einerseits die gegenseitige Nutzung von Zwischenergebnissen, andererseits sorgt sie dafür, dass die kürzesten Wege im Graphen in der Reihenfolge aufsteigender Gewichte gefunden werden.

Die gegenseitige Nutzung von Zwischenergebnissen beruht auf dem Konzept der optimalen Kanten. Eine Kante heißt optimal, wenn sie Teil eines kürzesten Weges ist. Für den Algorithmus gilt dann, dass zu jedem Zeitpunkt der Ausführung der nächstlängere zu berechnende kürzeste Weg entweder eine Kante ist oder aber nur aus optimalen Kanten zusammengesetzt ist.

Die Laufzeit des HiddenPath-Algorithmus hängt, ähnlich wie beim Algorithmus nach Dijkstra, stark von der zugrundeliegenden Implementierung der Prioritätswarteschlange ab. Bei Verwendung eines Fibonacci-Heaps liegt die Laufzeit in $O(nm^* + n^2 \lg n)$, wobei m^* die Anzahl der optimalen Kanten bezeichnet. In die Laufzeit geht also, verglichen mit dem n-fachen Dijkstra, nicht mehr die Anzahl aller Kanten ein, sondern nur noch die Anzahl der optimalen Kanten im Graph.

Wie bereits oben erwähnt, berechnet der HiddenPath-Algorithmus die kürzesten Wege im Graphen in der Reihenfolge aufsteigender Gewichte. Diese Eigenschaft ermöglicht wie beim Algorithmus nach Dijkstra die Definition einer beschränkten Variante, welche lediglich solche Wege berechnet, deren Gewicht einen vorgegebenen Schwellwert nicht überschreitet. Die Korrektheit der beschränkten Variante lässt sich analog zur Korrektheit des originalen Algorithmus beweisen [Karger et al., 1993], da

die wesentlichen Beweisschritte von der Beschränkung nicht betroffen sind.

Die beschränkte Variante des HiddenPath-Algorithmus wird in Abb. 2.4 dargestellt. Sie berechnet alle kürzesten Wege eines Graphen, die kürzer als der Schwellwert "limit" sind. Der Algorithmus basiert auf den folgenden drei Datenstrukturen:

- E_v^*: Für jeden Knoten v wird eine Liste optimaler Kanten E_v^* gespeichert. Die optimalen Kanten eines Knoten ermöglichen es, die Ausführung der Update()-Funktion in Zeile 15 auf diejenigen Kanten einzuschränken, die überhaupt Teil eines kürzesten Weges sein können.

- Queue: Die Prioritätswarteschlange Queue enthält eine Menge von Pfadkandidaten, die nach ihrem Gewicht aufsteigend sortiert sind. Insbesondere verfügt Queue über die Operation Queue.decreaseKey(p, w), welche dem Pfad p ein neues (kleineres) Gewicht w zuweist und somit zu einer Relozierung von p innerhalb von Queue führt.

- Paths: Die Pfadmatrix Paths enthält für jedes Knotenpaar u, v den derzeit kürzesten Pfad von u nach v. Ein Eintrag Paths$[u, v]$ besteht aus folgenden Komponenten:
  - Paths$[u, v]$.first: Erster Knoten auf dem Pfad $u \rightsquigarrow v$.
  - Paths$[u, v]$.weight: Gewicht des Pfads $u \rightsquigarrow v$.
  - Paths$[u, v]$.qptr: Verweis auf das Element welches $u \rightsquigarrow v$ in Queue repräsentiert.

Wie bereits oben diskutiert, hängt die Komplexität des Algorithmus wesentlich von der Implementierung der zugrundeliegenden Datenstrukturen ab. Auf eine detaillierte Diskussion des Algorithmus wird an dieser Stelle verzichtet und anstelle dessen auf Karger et al. [1993] verwiesen.

2.7.3 Bedeutung für den Entwurf

Die in diesem Abschnitt diskutierten Verfahren zur statischen Berechnung von Spreading Activation bzw. von kürzesten Wegen im Graph sind als Alternativen anzusehen bei der Interpretation der im ContextGraph kodierten semantischen Zusammenhänge. Beide Ansätze sind grundsätzlich auf das Szenario der kontextbasierten Suche anwendbar. Beim Entwurf des Systems wurde der kürzesten-Wege-Alternative der Vorrang eingeräumt, da durch die Notwendigkeit der Ausführung des Spreading-Activation-Algorithmus zur Anfragezeit und nach den Erkenntnissen von Rocha et al. [2004] ein Spreading-Activation-Algorithmus Probleme bei der Performanz erwarten lässt.

Zur Berechnung der kürzesten Wege wurde sowohl der beschränkte n-fache Dijkstra als auch der HiddenPath-Algorithmus implementiert und evaluiert, siehe dazu Abschnitt 5.4.

0 **Initialisierung**
1 Für alle Kanten $\langle u, v\rangle$ mit $\mathrm{dist}(\langle u, v\rangle) <$ limit:
2 Füge $\langle u, v\rangle$ in Queue ein: QElem := Queue.insert($\langle u, v\rangle$).
3 Setze das Kantengewicht im Paths-Eintrag: Paths$[u, v]$.weight := $\mathrm{dist}(\langle u, v\rangle)$.
4 Verknüpfe Paths-Eintrag mit Queue-Eintrag: Paths$[u, v]$.qptr := QElem.
5 Für alle v:
6 Initialisiere die optimalen Kanten: $\mathrm{E}_v^* := \emptyset$.
7
8 **Iterationen**
9 Solange Queue nicht leer und das Gewicht des ersten Elements kleiner limit:
10 1. Lösche den ersten Pfad $u \rightsquigarrow v$ aus Queue.
11 2. Falls $u \rightsquigarrow v$ eine Kante ist:
12 Markiere $\langle u, v\rangle$ als optimal für v: $\mathrm{E}_v^* := \mathrm{E}_v^* \cup \langle u, v\rangle$.
13 Für alle Knoten $w \in V$: Update(u, v, w).
14 3. Für alle für u optimalen Kanten $\langle t, u\rangle \in \mathrm{E}_u^*$:
15 Update(t, u, v).
16
17 **Procedure Update**(x, y, z)
18 Bestimme das Gewicht c des Pfads $x \rightsquigarrow y \rightsquigarrow z$:
19 $c :=$ Paths$[x, y]$.weight +Paths$[y, z]$.weight.
20 Falls $c < P[x, z]$.weight und $c <$ limit:
21 Paths$[x, z]$.weight := c .
22 Paths$[x, z]$.first := y .
23 Queue.decreaseKey$(x \rightsquigarrow z, c)$.

Abbildung 2.4: Beschränkter HiddenPath-Algorithmus.

2.8 Zusammenfassung

In den Abschnitten 2.1 bis 2.5 wurden Ansätze aus verschiedenen Bereichen der Suche vorgestellt. Zum Abschluss jedes Abschnitts wurde jeweils der Einfluss auf den in der vorliegenden Arbeit präsentierten Ansatz diskutiert.

Abschnitt 2.6 diskutiert die Grundlagen des betrieblichen Wissensmanagements. Der Nutzen einer kontextbasierten Intranet-Suchmaschine liegt demnach in der Unterstützung des Managements von Integrationswissen im Unternehmen.

In Abschnitt 2.7 schließlich wurden Spreading-Activation und die Berechnung der kürzesten Wege als alternative Berechnungsmodelle zur Bestimmung des inhaltlichen Zusammenhangs von Knoten eines Graphen diskutiert und argumentiert, warum in der vorliegenden Arbeit die Berechnung der kürzesten Wege einem Spreading-Activation-Algorithmus vorgezogen wird.

Der Rest dieser Zusammenfassung soll zum Abschluss eine Einordnung der verwandten Ansätze geben und somit zu einem besseren Überblick beitragen.

Die vorgestellten Ansätze können nach zwei Merkmalen kategorisiert werden.

- Daten. Bei der Art der durchsuchbaren Daten kann zwischen strukturierten

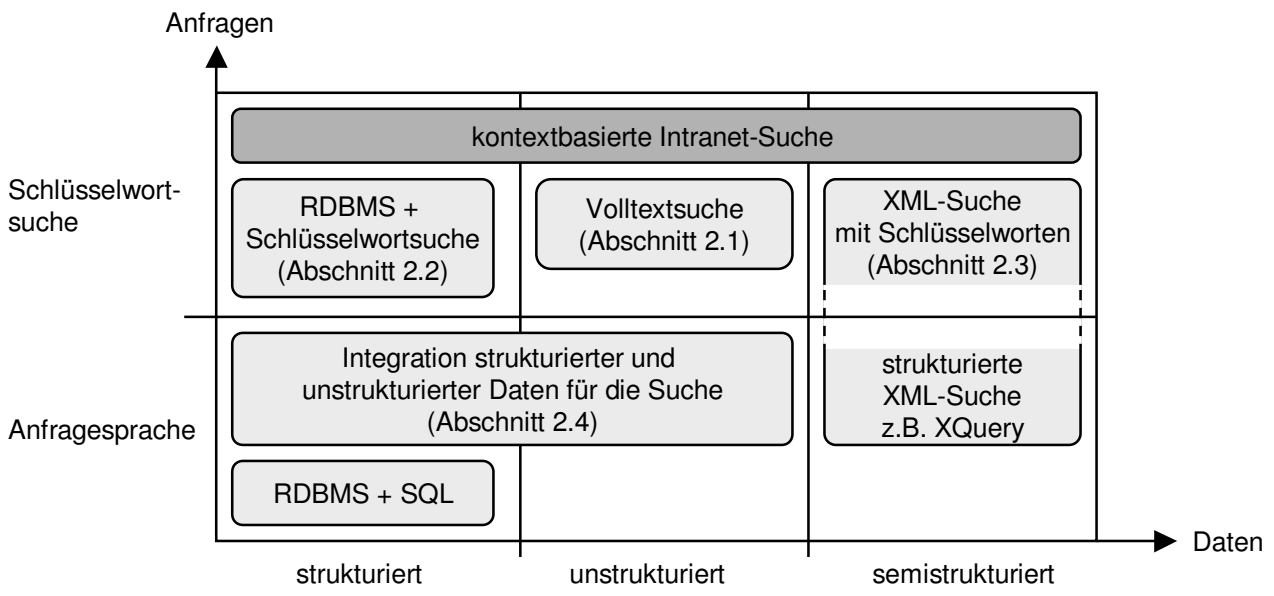


Abbildung 2.5: Einordnung der Ansätze bezüglich Daten und Anfragen.

Daten (etwa in DBMS), unstrukturierten Daten (d.h. Texte) und semistrukturierten Daten (z.B. XML) unterschieden werden.

- Anfragen. Bei der Form der Suchanfragen wird zwischen Anfragesprachen (z.B. SQL, XQuery) und Schlüsselwortsuche unterschieden.

Abbildung 2.5 gibt einen Überblick über die in den Abschnitten 2.1 bis 2.4 vorgestellten Ansätze. Von links unten ausgehend sind zunächst Lösungen mit Anfragesprachen auf strukturierten Daten zu nennen. In diese Kategorie fällt z.B. die große Menge relationaler Datenbanksysteme, auf welche über die Anfragesprache SQL zugegriffen werden kann. Die Eintragung dieser Systeme im Diagramm dient lediglich der Orientierung, eine entsprechende Diskussion liegt weit außerhalb des Rahmens der vorliegenden Arbeit. In Abschnitt 2.4 wurden Ansätze diskutiert, die per Anfragesprache auf strukturierte wie unstrukturierte Daten zugreifen. Die Schlüsselwortsuche auf relationalen Datenbanken wurde in Abschnitt 2.2 diskutiert, die Grundlagen zur Schlüsselwortsuche auf Dokumenten (Volltextsuche) in Abschnitt 2.1. Im Bereich der Suche auf semistrukturierten Daten wurden Ansätze zur Suche mittels einer Anfragesprache, wie z.B. XQuery, von der Betrachtung ausgenommen. Die Suche auf XML-Dokumenten mit Schlüsselwörtern hingegen wurde in Abschnitt 2.3 besprochen. Die im weiteren Verlauf der Arbeit diskutierte kontextbasierte Intranet-Suche schließlich erweitert die Schlüsselwortsuche auf Dokumenten um Kontextinformation aus strukturierten und semistrukturierten Datenquellen.

Im Bereich der semantischen Suche existiert ebenfalls eine große Vielfalt an Ansätzen. Zu ihrer Kategorisierung könnte Abbildung 2.5 um eine dritte Dimension erweitert werden, auf welcher die Verwendung von Wissensstrukturen aufgetragen wird. Die in Abschnitt 2.5 tatsächlich betrachteten Lösungen fallen dann allesamt in den Bereich der Wissensstruktur-gestützten Schlüsselwortsuche auf (teilweise anno-

tierten) unstrukturierten Daten.

Als Fazit lässt sich feststellen, dass keiner der in den Abschnitten 2.1 bis 2.5 vorgestellten Ansätze den in dieser Arbeit vorgestellten ersetzen kann. Beim Entwurf des Ansatzes der vorliegenden Arbeit wurde das Ziel verfolgt, vielversprechende Ideen aus allen angesprochenen Bereichen auf die kontextbasierte Intranet-Suche zu übertragen und symbiotisch zu verbinden.

3

Konzepte kontextbasierter Suche

In diesem Kapitel werden die zentralen Konzepte für die kontextbasierte Suche im Intranet vorgestellt. Zuerst wird der ContextGraph als Abstraktion zugrundeliegender Datenquellen eingeführt und die Abbildung relationaler Datenbanken auf den ContextGraph diskutiert. Die Kanten des ContextGraph werden mit einem asymmetrischen Maß für semantischen Abstand gewichtet, welches schließlich als Grundlage für die Berechnung des Kontexts von Dokumenten dient.

Im zweiten Teil des Kapitels werden Maße zur Bewertung von Ergebnissen kontextbasierter Suche entwickelt. Diese gliedern sich auf in Maße für die Relevanz, die Wichtigkeit sowie die Ähnlichkeit von Dokumenten.

3.1 Der ContextGraph

In diesem Abschnitt wird das für die vorliegende Arbeit zentrale Datenmodell vorgestellt, siehe dazu auch [Mangold et al., 2006a]. Es dient zur Abstraktion von Datenquellen und als Basis zur Verknüpfung strukturierter und unstrukturierter Daten für die Suche. Da im Datenmodell der Kontext von Dokumenten modelliert wird, erhält es den Namen *ContextGraph*.

3.1.1 Definition

Wie bereits in Kapitel 1 angesprochen, handelt es sich beim ContextGraph um ein graphenbasiertes Modell. Genauer gesagt handelt es sich um einen gerichteten, gewichteten, beschrifteten Graphen. Bevor eine Definition des ContextGraphs gegeben

wird, seien noch die folgenden Bezeichnungen festgelegt. Für den weiteren Verlauf der Arbeit sei

- $\mathbb{U}$ die Menge aller Uniform Ressource Identifier (URI),
- $\mathbb{N}_0$ die Menge der natürlichen Zahlen inklusive der 0,
- $\mathbb{T}$ die Menge aller im System verzeichneten Begriffe,
- $\mathbb{T}^*$ die Menge aller endlichen Konkatenationen $t_1\ t_2\ \ldots\ t_k$ von Begriffen aus $\mathbb{T}$, d.h. die Menge aller Texte und
- $\mathbb{D}$ die Menge aller im System verzeichneten Dokumente.

DEFINITION 3.1.1 (CONTEXTGRAPH)
Ein ContextGraph $CG = \langle V, E, \mathrm{vlabel}, \mathrm{docuri}, \mathrm{elabel}, \mathrm{dist}\rangle$ wird durch eine endliche Knotenmenge V, eine endliche Menge gerichteter Kanten E und die Funktionen $\mathrm{vlabel}, \mathrm{docuri}, \mathrm{elabel}$ und dist beschrieben. Dabei gelten die folgenden Eigenschaften:

- Die Knotenmenge $V = V_A \cup V_L$ setzt sich aus zwei disjunkten Mengen zusammen:
  - V_A ist die Menge aller Knoten ohne Bezeichner (A steht für "anonymous"). Die Teilmenge $V_D \subseteq V_A$ repräsentiert im ContextGraph die Menge der Dokumente $\mathbb{D}$. Ihre Elemente heißen Dokumentenknoten.
  - V_L ist die Menge aller beschrifteten Knoten; L steht für "label".
- Eine Kante $e = \langle u, v\rangle \in E$ verbindet die Knoten $u \in V$ und $v \in V$. Die Kanten in E sind *gerichtet*.
- Die Funktion $\mathrm{vlabel} : V_L \rightarrow \mathbb{T}^*$ ordnet jedem Knoten aus V_L eine textuelle Beschriftung zu.
- Die Funktion $\mathrm{elabel} : E \rightarrow \mathbb{T}^*$ ordnet jeder Kante eine textuelle Beschriftung zu.
- Die Funktion $\mathrm{docuri} : V_D \rightarrow \mathbb{U}$ ordnet jedem Dokumentenknoten eine URI zu.
- Die Funktion $\mathrm{dist} : E \rightarrow \mathbb{N}_0$ modelliert den gerichteten Abstand zwischen dem Start- und dem Endknoten einer Kante. Um die Darstellung zu vereinfachen, wird in dieser Arbeit der Wertebereich der $\mathrm{dist}()$-Funktion auf die natürlichen Zahlen incl. der Zahl 0 beschränkt. Prinzipiell sind hier auch die positiven reellen Zahlen $\mathbb{R}_0^+$ denkbar.

$\Diamond$

Die Knoten des ContextGraph repräsentieren Konzepte bzw. Entitäten der realen Welt. Zur Vereinfachung der Diskussion wird im Folgenden nicht immer explizit

zwischen Knoten des ContextGraphs und den durch sie repräsentierten Entitäten unterschieden, die jeweilige Bedeutung wird aus dem Kontext klar.

Wie in Graphen üblich, wird die Definition der Abstandsfunktion $\mathrm{dist}()$ von Kanten auf allgemeine Pfade des Graphen kanonisch erweitert:

DEFINITION 3.1.2 (ABSTANDSFUNKTION ENTLANG EINES PFADES)
Sei $p = \langle e_1, \ldots, e_n \rangle$ mit $e_1 = \langle v_0, v_1 \rangle, \ldots, e_n = \langle v_{n-1}, v_n \rangle \in E$ ein beliebiger zusammenhängender Pfad der Länge $n > 0$ und $e_0 = \langle u, v_0 \rangle \in E$ eine Kante, die auf dem Startknoten v_0 von p endet. Dann ist der gerichtete Abstand zwischen den Knoten u und v_n entlang des nach vorne verlängerten Pfads $p' = \langle e_0, e_1, e_2, \ldots, e_n \rangle$ definiert als:

$$\mathrm{dist}(p') = \begin{cases} \mathrm{dist}(e_0) + \mathrm{dist}(e_1) & \text{falls } n = 1 \\ \mathrm{dist}(e_0) + \mathrm{dist}(p) & \text{sonst.} \end{cases}$$

$\Diamond$

Mit dieser Definition lässt sich der Abstand zwischen beliebigen Knoten im Graphen folgendermaßen definieren:

DEFINITION 3.1.3 (ABSTANDSFUNKTION ZWISCHEN KNOTEN)
Seien $u \in V$ und $v \in V$ beliebige Knoten und sei P die Menge der Pfade von u nach v. Dann ist der gerichtete Abstand zwischen u und v definiert als:

$$\mathrm{dist}(u, v) = \begin{cases} \min_{p \in P}(\mathrm{dist}(p)) & \text{falls } |P| > 0 \\ \infty & \text{falls } P = \emptyset. \end{cases}$$

$\Diamond$

Diskussion Die hier beschriebene Definition des ContextGraphs ist vergleichbar mit graphenbasierten Modellen aus den Bereichen der semistrukturierten Datenmodelle, der Datenbankgraphen bei der Schlüsselwortsuche in relationalen Datenbanken und der Wissensstrukturen der semantischen Dokumentensuche. Keiner der im Folgenden angesprochenen Ansätze verfügt über Mechanismen zur Integration Graph-externer Dokumente, wie dies beim ContextGraph durch Dokumentenknoten und die Funktion $\mathrm{docuri}()$ realisiert ist.

Aus dem Bereich der semistrukturierten Datenmodelle sei verwiesen auf das Object Exchange Model (OEM) [Papakonstantinou et al., 1995], das von Buneman et al. [1996] vorgestellte, baumbasierte Datenmodell und auf das graphenbasierte Datenmodell, welches in Strudel [Fernandez et al., 1998] zum Einsatz kommt. Im Gegensatz zu den ursprünglichen Versionen dieser Datenmodelle verfügt der ContextGraph über Kantengewichte; eine erweiterte Version von OEM wird in [Goldman et al., 1998] zum Zwecke der Kontextsuche in OEM-Graphen ebenfalls um ein Abstandsmaß auf Kanten erweitert.

Im Bereich der Schlüsselwortsuche in relationalen Datenbanken wurde in Abschnitt 2.2.1 die Hyperbase des DataSpot-Systems sowie in Abschnitt 2.2.3 der Datenbankgraph des BANKS-Systems vorgestellt. Beide sind in ihrer Struktur in etwa vergleichbar mit dem hier definierten ContextGraph.

Da Wissensstrukturen häufig in graphartigen Modellen repräsentiert werden, nutzen im Bereich der semantischen Suche viele in Abschnitt 2.5.3 vorgestellten Ansätze Graphen. Oftmals wird dort jedoch, wie von Merkmal "Struktur der Ontologie" in Abschnitt 2.5.2 reflektiert, die Bedeutung der Kantenbeschriftung zur Bearbeitung von Anfragen herangezogen. Ein Maß für den semantischen Abstand zwischen Knoten ist dann nicht erforderlich.

Im folgenden wird der ContextGraph als Abstraktionsmechanismus eingesetzt, um unabhängig von einzelnen Unternehmensinformationssystemen den Kontext von Dokumenten zu ermitteln.

3.1.2 Einbettung relationaler Datenbanken

Im vorigen Abschnitt wurde der ContextGraph formal definiert. Da sich die vorliegende Arbeit vorwiegend mit der Einbettung von Daten aus RDBMS beschäftigt, soll in diesem Abschnitt auf ebendiese Einbettung eingegangen werden. Die Einbettung von graphbasierten Wissensmodellen oder semistrukturierten Daten in den ContextGraph kann analog zu dieser Abbildung leicht hergeleitet werden.

Um die Abbildung einer relationalen Datenbank (RDB) auf den ContextGraph beschreiben zu können, wird im Folgenden eine formale Definition einer RDB gegeben, vgl. [Date, 1982]. Die hier vorgestellte Definition beschränkt sich auf die Teile der RDB, die für die Abbildung relevant sind. Hier nicht vorgestellte Elemente (wie z.B. Schlüsseleigenschaften, virtuelle Tabellen, relationale Operatoren, Constraints, etc.) werden nicht im ContextGraph repräsentiert.

Im Folgenden repräsentieren Großbuchstaben jeweils Mengen, Kleinbuchstaben die zugehörigen Elemente. Die Schreibweise $a.b$ wird verwendet, um den Bestandteil b des Elements a zu referenzieren.

DEFINITION 3.1.4 (RELATIONALE DATENBANK)

- Eine relationale Datenbank $rdb = \langle W, R, F \rangle$ wird spezifiziert durch eine Menge W von Wertebereichen (engl. Domains), eine Menge R von Relationen und eine Menge F von Fremdschlüsselbeziehungen.

- Ein Wertebereich $w = \langle n, V \rangle$ wird durch einen Namen n und eine Menge V von Werten beschrieben.

- Eine Relation $r = \langle n, A, T \rangle$ besteht aus einem Namen n, einer Menge A von Attributen und einer Menge T von Tupeln.

- Ein Attribut $a = \langle n, w.n \rangle$ wird durch einen Namen n und einen Wertebereichsnamen $w.n$ definiert, der dem Attribut a den Wertebereich w zuweist.

- Ein Tupel $t = S$ besteht aus einer Menge S von Attributwerten, in der jedes Element $s = \langle a.n, w.v \rangle$ einen Attributnamen $a.n$ und einen Wert $w.v$ des durch den Attributnamen festgelegten Wertebereichs enthält.

- Eine Fremdschlüsselbeziehung $f = \langle K, L \rangle$ besteht aus zwei gleichgroßen, geordneten Mengen von Attributnamen $K = \langle k_1 \dots k_n \rangle$ und $L = \langle l_1 \dots l_n \rangle$. Die Menge K bzw. L ist dabei jeweils Teilmenge der Attributnamen einer Relation $r_1 = \langle n_1, A_1, T_1 \rangle$ bzw. $r_2 = \langle n_2, A_2, T_2 \rangle$. Dann gilt für alle Tupel $t_1 = S_1 \in T_1$: für alle Attributwerte $s_1 \in S_1$, für die ein x existiert mit $s_1.a.n = k_x$, existiert ein $t_2 = S_2 \in T_2$ mit einem $s_2 \in S_2$, sodass $s_2.a.n = l_x$ und $s_1.w.v = s_2.w.v$. Das Tupel t_2 erfüllt somit die Fremdschlüsselbeziehung für t_1. $\Diamond$

Im Folgenden wird davon ausgegangen, dass die Dokumente in $\mathbb{D}$ in einer RDB als Tupel modelliert sind. Um die Betrachtung zu vereinfachen, wird weiterhin angenommen, dass jedes solche Tupel über ein Attribut verfügt, welches den Uniform Ressource Identifier (URI) des Dokuments beinhaltet. Über das URI-Attribut hinaus kann ein solches Tupel beliebig viele Attribute besitzen. Eine relationale Datenbank kann dann wie folgt in einen ContextGraph transformiert werden:

1. Relationen: Jede Relation $r \in R$ wird abgebildet auf einen Knoten $v_r \in V_L$, der mit dem Namen der Relation beschriftet ist, d.h. die Funktion $vlabel()$ wird definiert als $vlabel(v_r) = r.n$.

2. Tupel: Jedes Tupel t einer Relation r wird abgebildet auf einen anonymen Knoten $v_t \in V_A$. Zusätzlich wird v_t mit v_r durch zwei Kanten $e_1 = \langle v_t, v_r \rangle$ und $e_2 = \langle v_r, v_t \rangle$ verbunden, für die $elabel()$ definiert wird als $elabel(e_1) =$ "is_a" bzw. $elabel(e_2) =$ "has_instance". Für Tupel, welche Dokumente repräsentieren, gilt $v_t \in V_D$. Der Funktionswert von $docuri(v_t)$ ergibt sich dann aus dem Attributwert von t, welcher die URI des Dokuments enthält.

3. Attributwerte: Jeder Attributwert s eines Tupels t (außer solchen, die URIs von Dokumenten beschreiben) wird abgebildet auf einen Knoten $v_s \in V_L$, der mit dem jeweiligen Wert beschriftet ist, d.h. $vlabel(v_s) = w.v$. Zwei Kanten $e_1 = \langle v_t, v_s \rangle$ und $e_2 = \langle v_s, v_t \rangle$ verbinden das Tupel mit dem Attributwert. Die Vorwärtskante ist dabei mit dem Namen des Attributs beschriftet, d.h. $elabel(e_1) = a.n$, die Rückwärtskante zeigt die inverse Beziehung an: $elabel(e_2) =$ "inv(" $a.n$ ")".

4. Fremdschlüssel: Fremdschlüsselbeziehungen werden auf Kanten in E abgebildet. Für alle Paare (t_1, t_2) von Tupeln, die nach Definition 3.1.4 die Fremdschlüsselbeziehung erfüllen, werden zwei Kanten $e_1 = \langle v_{t_1}, v_{t_2} \rangle$ und $e_2 =$

$\langle v_{t_2}, v_{t_1} \rangle$ erzeugt. Den Kantenbeschriftungen $elabel(e_1)$ und $elabel(e_2)$ wird die leere Zeichenkette zugewiesen.

5. M:N-Beziehungen: Bei der relationalen Modellierung einer M:N-Beziehung zwischen zwei Entitätenmengen der realen Welt wird eine Hilfsrelation eingeführt [Kemper und Eickler, 2004, Kap. 3]. Die Attribute dieser Hilfsrelation verweisen als Fremdschlüssel auf die Tupel der Relationen der beiden Entitätenmengen. Da im ContextGraph M:N-Beziehungen direkt modelliert werden können, findet diese Hilfsrelation keine Abbildung im ContextGraph. Seien $r_1 = \langle n_1, A_1, T_1 \rangle, r_2 = \langle n_2, A_2, T_2 \rangle \in R$ zwei Relationen, die über die Hilfsrelation $r_h = \langle n_h, A_h, T_h \rangle \in R$ und über die Fremdschlüsselbeziehungen $f_1 = \langle S_1, D_1 \rangle, f_2 = \langle S_2, D_2 \rangle \in R$ verbunden sind mit $S_1 \cup S_2 = A_h$ und $D_1 \subseteq A_1$ bzw. $D_2 \subseteq A_2$. Nach (4) werden dann für jedes Tupel $t \in T_h$ ein Knoten $n_t \in V_A$ sowie die vier Kanten $e_1^{fw} = \langle v_t,\ v_1 \rangle,\ e_1^{bw} = \langle v_1, v_t \rangle,\ e_2^{fw} = \langle v_t, v_2 \rangle, e_2^{bw} = \langle v_2, v_t \rangle$ eingefügt, die v_t mit den beiden Knoten v_1 und v_2 verbinden, die die per f_1 und f_2 designierten Tupel aus T_1 und T_2 repräsentieren. Diese Modellierung lässt sich vereinfachen, indem die Knoten v_1 und v_2 direkt durch zwei Kanten $e^{fw} = \langle v_1, v_2 \rangle,\ e^{bw} = \langle v_2, v_1 \rangle$ ohne Zwischenknoten verbunden werden.

Die Zuweisung der Kantenbeschriftungen, wie durch die Funktion $elabel()$ modelliert, dient lediglich der Verständlichkeit des Graphen für menschliche Benutzer und hat keinen Einfluss auf die in dieser Arbeit vorgestellten Algorithmen zur Auswertung der im ContextGraph modellierten Information. Daher können die hier zugeordneten Werte selbstverständlich manuell überschrieben werden. Die Definition der $dist()$-Funktion kann auf verschiedene Arten durchgeführt werden und wird in Abschnitt 3.1.3 beschrieben.

Zur Illustration der Abbildung einer RDB auf einen ContextGraph wird in Abbildung 3.1 der Inhalt von vier Relationen beispielhaft transformiert.

Der Ausschnitt des Produktionsplanungssystems (PPS) beinhaltet die Relationen Maschine, Teil und Produktion. Die Relation Produktion übernimmt dabei die Funktion einer Hilfsrelation, die mittels Fremdschlüsselbeziehungen (FS) die M:N-Beziehung zwischen Maschine und Teil modelliert. Die Relation Fehlerbericht liegt im Dokumentenmanagementsystem (DMS). Die Attribute Teil-ID und Maschine-ID bilden dabei Fremdschlüsselbeziehungen zum Attribut ID der Relation Teil bzw. zum Attribut ID der Relation Maschine.

Bei der Abbildung in den ContextGraph werden die Relationen Maschine, Teil, und Fehlerbericht auf gleichnamige Knoten abgebildet. Die Relation Produktion entfällt, wie oben beschrieben, da sie lediglich die M:N-Beziehung zwischen den Relationen Maschine und Teil modelliert. Die Tupel der ersten beiden Relationen werden auf anonyme Knoten abgebildet. Das einzige Fehlerbericht-Tupel wird zum Dokumentenknoten. Die Attribute aller Knoten werden jeweils zu beschrifteten Knoten

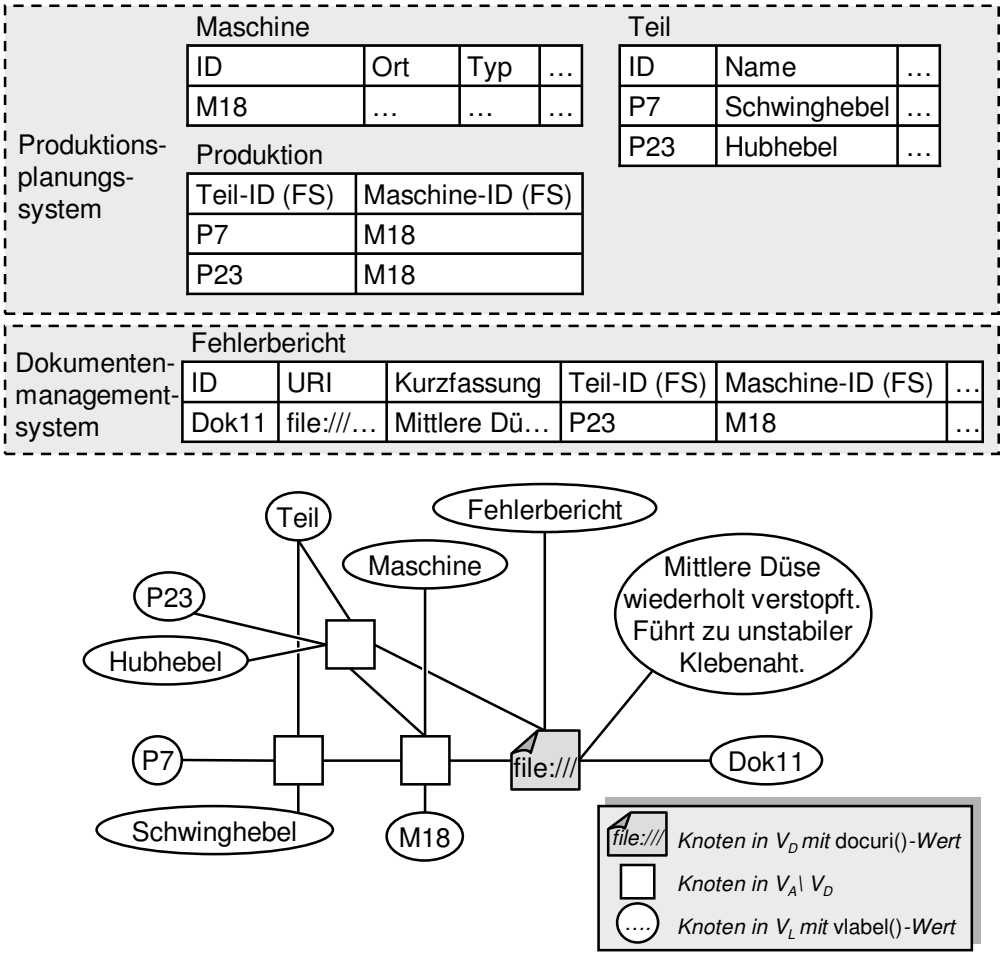


Abbildung 3.1: Beispielhafte Abbildung einer RDB auf einen ContextGraph.

mit Ausnahme der Werte des URI-Attributs der Relation Fehlerbericht, welche im Dokumentenknoten als Wert der Funktion $docuri()$ hinterlegt werden.

Der Übersichtlichkeit halber werden in der Abbildung je zwei sich entgegen gerichtete Kanten als eine ungerichtete Kante dargestellt; die Beschriftungen und Gewichtungen der Kanten werden aus demselben Grund ebenfalls nicht dargestellt.

Diskussion Neben der hier vorgestellten Abbildung einer RDB auf den ContextGraph sind noch andere, alternative Abbildungen denkbar.

Anstatt jeden Attributwert auf einen eigenen Knoten abzubilden, kann ebenfalls jedes Tupel zusammen mit allen Attributwerten auf einen einzigen Knoten projiziert werden. Dieser Ansatz wurde z.B. bei dem in Abschnitt 2.2.3 vorgestellten BANKS System verfolgt [Bhalotia et al., 2002]. Einerseits führt dies zu einer wesentlich kompakteren Graphenstruktur, da Kanten ausschließlich durch Fremdschlüsselbeziehungen gebildet werden. Andererseits geht dadurch die Möglichkeit verloren, verschiedene Attributwerte eines Tupels über verschiedene Kantengewichte (siehe Abschnitt 3.1.3) mit dem Tupelknoten verbinden zu können.

Um die Größe des ContextGraph zu beschränken, können, entsprechend der je-

weiligen Domäne, für die Dokumentenrecherche unwichtige Teile der relationalen Datenbank von der Abbildung in den ContextGraph ausgenommen werden. Dies kann einerseits gesamte Tabellen und andererseits einzelne Attribute betreffen. Aus logischer Sicht ist dann der ContextGraph auf einer Menge von Views auf der Datenbank definiert.

Analog zur Einbettung relationaler Datenbanken lässt sich leicht der Import von Daten aus anderen Datenmodellen in den ContextGraph herleiten. Beim Import von im XML-Format gespeicherten Daten würden XML-Elemente etwa auf Knoten und deren Hierarchiebeziehungen auf Kanten des ContextGraphs abgebildet werden. Der Import von Daten aus graphbasierten Datenmodellen, wie etwa in [Mangold et al., 2003; Mangold und Mitschang, 2003] vorgestellt, oder aus Ontologien, die bspw. im RDF-Format vorliegen, müssen lediglich Knoten auf Knoten und Kanten auf Kanten abgebildet werden. Mögliche im Datenmodell enthaltene Strukturbeschränkungen (engl.: Constraints) müssen – wie bereits oben erläutert – im ContextGraph nicht berücksichtigt zu werden, da die Suchmaschine nur lesend auf ihn zugreift.

3.1.3 Semantischer Abstand und Kontext

Mit der in Definition 3.1.3 eingeführten Abstandsfunktion zwischen zwei Knoten $\mathrm{dist}()$ wird bei der Modellierung eines ContextGraphs der inhaltliche Abstand zwischen Entitäten der realen Welt repräsentiert. So kann bspw. festgelegt werden, dass der inhaltliche Abstand eines Dokuments zu der Abteilung, in der es entstand, größer ist als zu seinem Autor. Da dieser inhaltliche Abstand in der Literatur häufig mit "semantischer" Abstand (engl.: semantic distance) bezeichnet wird, z.B. [Budanitsky, 1999], wird diese Bezeichnung für die vorliegende Arbeit übernommen.

3.1.3.1 Asymmetrie des semantischen Abstands

Auf den ersten Blick erscheint der semantische Abstand als ein symmetrisches Maß, d.h. für zwei sich entgegengerichtete Kanten $\langle u, v\rangle, \langle v, u\rangle \in E$ müsste gelten $\mathrm{dist}(\langle u, v\rangle) = \mathrm{dist}(\langle v, u\rangle)$. Betrachtet man ein Dokument und seinen Autor, so hat der Autor intuitiverweise den gleichen Abstand von seinem Dokument, wie das Dokument vom Autor. Durch die oben eingeführte Additivitätseigenschaft des Abstands entlang eines Pfades ergibt sich jedoch folgende Problematik, die auch an verschiedenen Stellen in der Literatur diskutiert wird, z.B. [Hulgeri et al., 2001; Bhalotia et al., 2002; Crestani, 1997].

Ist auf einem Graphen eine symmetrische Abstandsfunktion definiert, so besteht folgendes Problem: Oftmals hängt ein Knoten mit hohem Kantengrad inhaltlich mit seinen direkt benachbarten Knoten zusammen. Die Additivität des Abstands führt jedoch dazu, dass somit alle diese benachbarten Knoten ebenfalls als inhaltlich zusammenhängend modelliert sind.

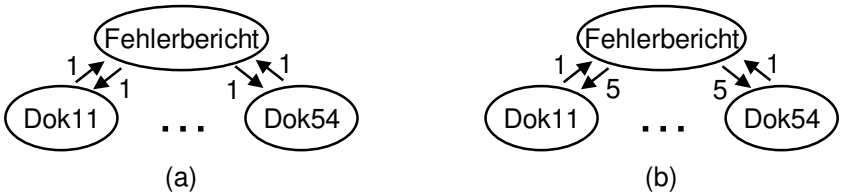


Abbildung 3.2: Semantischer Abstand. (a) Symmetrische Definition. (b) Asymmetrische Definition.

Werden im ContextGraph eines Unternehmens bspw. Fehlerberichte als Knoten abgebildet, so wird jeder Fehlerberichtsknoten mit dem Knoten verbunden sein, der die Bezeichnung "Fehlerbericht" trägt und somit die Rolle der einzelnen Fehlerberichtsknoten beschreibt. Diese Situation ist in Abbildung 3.2 skizziert.

Die symmetrische Definition des semantischen Abstands, wie in Abb. 3.2(a) abgebildet, assoziiert die Knoten Dok11 und Dok54 mit dem Knoten Fehlerbericht mit Abstand eins (d.h. $\mathrm{dist}(\text{Dok11}, \text{Fehlerbericht}) = \mathrm{dist}(\text{Fehlerbericht}, \text{Dok11}) = \mathrm{dist}(\text{Dok54}, \text{Fehlerbericht}) = \mathrm{dist}(\text{Fehlerbericht}, \text{Dok54}) = 1$). Dadurch ergibt sich einerseits der erwünschte Effekt, dass die Fehlerberichte einen kleinen Abstand zum Begriff "Fehlerbericht" haben. Andererseits gilt jedoch $\mathrm{dist}(\text{Dok54}, \text{Dok11}) = 2$. Diese Assoziierung kann durchaus unerwünscht sein, da so alle Fehlerberichte des Unternehmens als mit Abstand zwei inhaltlich zusammenhängend gewertet werden, obwohl zwischen den Fehlerberichten inhaltlich kein Zusammenhang besteht.

Die Lösung des Dilemmas ist in Abb. 3.2(b) abgebildet. Die Einführung eines gerichteten Abstands ermöglicht die korrekte Modellierung der Situation. Die Knoten Dok11 und Dok54 haben jeweils kleinen Abstand zum Knoten Fehlerbericht. Dadurch, dass der Knoten Fehlerbericht großen Abstand zu den Knoten Dok11 und Dok54 hat, haben Dok11 und Dok54 untereinander ebenfalls großen Abstand (im Beispiel ist $\mathrm{dist}(\text{Doc11}, \text{Doc54}) = \mathrm{dist}(\text{Dok11}, \text{Dok54}) = 6$).

3.1.3.2 Bestimmung des semantischen Abstands

In der Literatur findet sich eine Vielzahl von Vorschlägen zur Festlegung des semantischen Abstands, für eine Übersicht siehe [Budanitsky, 1999; Budanitsky und Hirst, 2001]. Viele dieser Ansätze konzentrieren sich jedoch auf eine Definition des semantischen Abstands zwischen Konzepten in elektronischen Wörterbüchern oder Lexika, wie bspw. WordNet. Im Unterschied zu dem in dieser Arbeit vorgestellten ContextGraph, in welchem die Bedeutung der Kanten a priori unbestimmt ist, bieten elektronische Lexika meist umfassende Hierarchien von Hypernymen (d.h. Entitätsverknüpfungen entlang der "ist-ein"-Beziehung) oder Meronymen (d.h. Entitätsverknüpfungen entlang der "ist-Teil-von"-Beziehung).

Im Folgenden soll kurz auf die grundlegenden Ideen zur Festlegung des semantischen Abstands eingegangen werden, welche keine Hierarchiebeziehung zwischen Konzepten zugrunde legen. Alle Ansätze haben gemeinsam, dass zunächst der se-

mantische Abstand aller Knotenpaare festgelegt wird, die durch eine Kante verbunden sind. Die Verallgemeinerung auf den semantischen Abstand zwischen beliebigen Knoten wird dann, wie in Definition 3.1.3 beschrieben, vorgenommen.

- Konstant. Der naheliegendste Ansatz zur Bestimmung der Abstände weist jeder Kante den gleichen Abstand zu. Da dies eine symmetrische Abstandsdefinition impliziert, handelt es sich nach obiger Diskussion um ein wenig taugliches Modell.

- Manuell. Bei der manuellen Bestimmung der Kantengewichte und damit des semantischen Abstands übernimmt ein Experte die Definition. Dieser Ansatz wird z.B. in [Balmin et al., 2004] verfolgt, ist aber offensichtlich nur für kleine Graphen geeignet oder für Graphen mit festgelegtem Schema, bei denen die Kantengewichte nicht individuell, sondern anhand des Schemas zugewiesen werden können.

- Kantenausgangsgrad. Ein mehrfach vorgestellter Ansatz (z.B. von Bhalotia et al. [2002] im BANKS-System) berechnet den gerichteten semantischen Abstand proportional zum Kantenausgangsgrad des Startknoten. Da der Graph verschiedene Kantentypen beinhaltet, ist dabei jeweils nur der Ausgangsgrad von Kanten desselben Typs relevant. Je mehr Kanten desselben Typs einen Knoten verlassen, desto höher ist daher der durch sie repräsentierte semantische Abstand. Im Beispiel aus Abb. 3.2(b) bedeutet dies, dass die angegebenen Abstände zutreffen, falls der "Fehlerbericht"-Knoten mit insgesamt fünf Dokumentenknoten verbunden ist. Auch das Konzept des Kantenausgangsgrads wurde mehrfach weiterentwickelt, z.B. [Rocha et al., 2004].

Für die vorliegende Arbeit sind grundsätzlich alle Arten von Kantengewichtungen denkbar. Aufgrund der erforderlichen Assymetrie lässt jedoch der Ansatz der konstanten Zuweisung ein wenig befriedigendes Ergebnis vermuten. Die manuelle Zuweisung ist hingegen durchaus in Betracht zu ziehen. Da ein aus einer relationalen Datenbank generierter ContextGraph dem Schema der Datenbank entspricht, können in diesem Falle die Kantengewichte gemäß dieses Schemas zugewiesen werden.

3.1.3.3 Der Kontext

Basierend auf dem Maß für semantischen Abstand, lässt sich über den ContextGraph der Kontext eines Dokuments wie folgt definieren:

DEFINITION 3.1.5 (KONTEXT EINES KNOTEN)
Sei $CG = (V, E, \mathrm{vlabel}, \mathrm{docuri}, \mathrm{elabel}, \mathrm{dist})$ ein ContextGraph nach Definition 3.1.1, wobei die Definitionsmenge von $\mathrm{dist}()$ nach Definition 3.1.3 auf beliebige Knotenpaare erweitert ist, $cRange \in \mathbb{N}_0$ eine positive ganze Zahl und $n \in V$ ein beliebiger

Knoten. Dann wird mit $\text{context}_{cRange}(n)$ die Menge an Knoten beschrieben, deren Abstand von n kleiner als $cRange$ ist:

$$\text{context}_{cRange}(n) = \{v \in V \mid \text{dist}(n, v) < cRange\}$$

$\Diamond$

Um spätere Berechnungen zu erleichtern, wird aus dem semantischen Abstand ein Maß für die semantische Nähe zwischen Knoten des Graphen abgeleitet.

DEFINITION 3.1.6 (SEMANTISCHE NÄHE)
Seien CG und $cRange$ wie in Definition 3.1.5. Dann wird die (gerichtete) semantische Nähe zweier Knoten $n_1, n_2 \in N$ folgendermaßen definiert:

$$\text{sim}_{cRange}(n_1, n_2) = \begin{cases} cRange - \text{dist}(n_1, n_2) & \text{, falls } \text{dist}(n_1, n_2) < cRange \\ 0 & \text{, sonst.} \end{cases}$$

$\Diamond$

Aus Definition 3.1.6 folgt direkt, dass die semantische Nähe eines Knoten $n_1 \in V$ zu einem Knoten $n_2 \in V$ genau dann 0 ist, wenn n_2 nicht im Kontext von n_1 liegt. Je kleiner der semantische Abstand von n_1 zu n_2, desto größer ist die semantische Nähe. Die semantische Nähe von n_1 zu sich selbst nimmt den maximalen Wert $cRange$ an.

3.1.3.4 Berechnung der kürzesten Wege

Zur Berechnung des Kontexts von Dokumenten im ContextGraph muss nach Definition 3.1.5 (Kontext eines Knoten) der semantische Abstand dist zwischen dem Dokumentenknoten und den Knoten seiner Nachbarschaft berechnet werden. Nach Definition 3.1.3 (semantischer Abstand zwischen Knoten) berechnet sich dieser als der kürzeste Weg zwischen zwei Knoten. Für die Berechnung der Kontexte von Dokumenten kann daher einer der in Abschnitt 2.7.2 vorgestellten Algorithmen zur Berechnung aller kürzesten Wege im Graph eingesetzt werden.

Die in Abschnitt 2.7.2 vorgestellten Algorithmen zur beschränkten Kürzesten-Wege-Suche sind jeweils für allgemeine Graphen anwendbar. Zur Optimierung der Laufzeit der Algorithmen können für das vorliegende Szenario zwei offensichtliche Einschränkungen des allgemeinen Falles vorgenommen werden.

- Einschränkung auf Dokumentenknoten. Es müssen nicht alle kürzesten Wege im Graph berechnet werden, da nur der Kontext von Dokumentenknoten von Interesse ist.

- k-Partitheit. Da der ContextGraph nach Abschnitt 3.1.2 durch Transformation aus einer relationalen Datenbank entsteht, erfüllt er die Anforderungen an k-partite Graphen, die eine Teilmenge allgemeiner Graphen bilden.

Die Einschränkung der Kontextbestimmung auf Dokumentenknoten ist offensichtlich und kann beim n-fachen Dijkstra leicht umgesetzt werden, indem nur Dokumentenknoten als Startknoten verwendet werden. Diese einfach zu implementierende Optimierung wurde im Prototypen umgesetzt.

Die k-Partitheitseigenschaft des ContextGraphs ist weniger leicht zu behandeln. Da in der Literatur keine Aussagen gefunden werden konnten, ob sich k-Partitheit nutzbringend für die Berechnung der kürzesten Wege im Graph ausnutzen lässt, wurden diesbezügliche Untersuchungen angestellt, die im Folgenden kurz zusammengefasst werden.

Zu diesem Zweck wird zunächst eine Definition bipartiter und k-partiter Graphen gegeben. Anschließend wird gezeigt, dass kein Algorithmus mit glatter Komplexitätsfunktion existiert, der die k-Partitheit eines Graphen zu einer asymptotischen – d.h. im Sinne der O-Notation signifikanten – Laufzeitreduktion verwenden kann.

DEFINITION 3.1.7 (BIPARTITE GRAPHEN)
Sei $G = (V, E)$ ein Graph mit Knotenmenge V und Kantenmenge E. Wir nennen G *bipartit*, falls zwei disjunkte Teilmengen $V_1, V_2 \subset V$ existieren, für die jeweils gilt, dass keine Kante zwei Knoten derselben Teilmenge verbindet, d.h.:

$$\text{Für alle } \langle u, v\rangle \in E: \text{ entweder } u \in V_1, v \in V_2 \text{ oder } u \in V_2, v \in V_1.$$

$\Diamond$

Gegenüber bipartiten Graphen werden k-partite Graphen durch eine Abschwächung der Partitionierungseigenschaft definiert:

DEFINITION 3.1.8 (k-PARTITE GRAPHEN)
Sei $G = (V, E)$ ein Graph mit Knotenmenge V und Kantenmenge E. Wir nennen G *k-partit*, falls k disjunkte Teilmengen $V_1, \ldots, V_k \subset V$ existieren, für die jeweils gilt, dass keine Kante zwei Knoten derselben Teilmenge verbindet, d.h.:

$$\text{Für alle } \langle u, v\rangle \in E: u \in V_i, v \in V_j \Rightarrow i \neq j.$$

$\Diamond$

Wie sich leicht anhand der Definition der Abbildung einer relationalen Datenbank auf einen ContextGraph nachvollziehen lässt, handelt es sich bei einem derart erzeugten ContextGraph um einen k-partiten Graphen, vgl. Abschnitt 3.1.2. Folgende Knotenmengen können jeweils zu einer Partition zusammengefasst werden, da keine zwei Knoten aus ihnen direkt miteinander verbunden sind:

- Alle Knoten, die die Tupel einer Relation repräsentieren.
- Alle Knoten, die die Attributwerte einer Relation repräsentieren.
- Alle Knoten, die eine Relation repräsentieren.

Für ein Relationenschema mit n Tabellen entsteht demnach ein $(2n+1)$-partiter ContextGraph. Einen Sonderfall bildet das Szenario, in dem eine Relation per Fremdschlüssel auf sich selbst verweist und somit Tupelknoten einer Relation direkt miteinander verbunden sind. Ohne die semantischen Abstände im Graphen zu beeinträchtigen, kann dieser Situation mit einer Menge von Hilfsknoten begegnet werden, die "in der Mitte" jeder durch eine solche Fremdschlüsselbeziehung eingeführten Kante eingefügt werden, d.h. für jede solche Kante $\langle u, v\rangle$ wird ein Knoten x eingefügt und $\langle u, v\rangle$ durch die Kanten $\langle u, x\rangle$ und $\langle x, v\rangle$ ersetzt. Dabei muss gelten, dass $\mathrm{dist}(\langle u, x\rangle) + \mathrm{dist}(\langle x, v\rangle) = \mathrm{dist}(\langle u, v\rangle)$ gilt. Diese so eingeführte Hilfsknotenmenge bildet dann eine eigene Partition.

Zur Formulierung einer Aussage über die Laufzeit von APSP-Algorithmen wird noch folgende Definition benötigt, die sich am Lehrbuch von Brassard und Bratley [1988] orientiert.

DEFINITION 3.1.9 (GLATTE FUNKTION, GLATTER ALGORITHMUS)
Eine Funktion $f : \mathbb{N} \to \mathbb{R}$ heißt *glatt* (engl.: smooth), falls

1. es ein $n_0 \in \mathbb{N}$ gibt, ab dem f monoton steigt, d.h. $(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)[f(n) \leq f(n+1)]$, und
2. es ein $b \in \mathbb{N}$ gibt mit $b \geq 2$, sodass $f(bn) \in O(f(n))$.

Ein Algorithmus A heißt glatt, wenn seine Laufzeit in $\Theta(f)$ liegt für eine glatte Funktion f. $\Diamond$

Tatsächlich folgt aus der Definition der O-Notation, dass für eine glatte Funktion die zweite Bedingung für alle $b \in \mathbb{N}$ gilt. Beispiele für glatte Funktionen sind $n, n^2, \log n, n \log n, \sqrt{n}$ sowie alle Polynome, Logarithmusfunktionen, Wurzelfunktionen und daraus zusammengesetzte Funktionen. In die Klasse der nicht-glatten Funktionen fällt z.B. 2^n oder abschnittsweise definierte Funktionen, die stellenweise sehr steil ansteigen. Für eine solche Funktion gilt dann $(\forall c)(\forall n_0)(\exists n \geq n_0)[f(bn) > c \cdot f(n)]$. Für die folgende Diskussion wird die Definition der Glattheit kanonisch erweitert auf Funktionen $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, die von zwei Variablen abhängen. Damit sind die Voraussetzungen geschaffen für folgendes Lemma.

LEMMA 3.1.10 (KÜRZESTE WEGE AUF BIPARTITEN GRAPHEN)
Sei A ein Algorithmus, der das APSP-Problem auf beliebigen Graphen optimal, d.h. in minimaler Laufzeit, löst. Dann existiert kein glatter Algorithmus A', der das APSP-Problem auf bipartiten Graphen in asymptotisch kürzerer Laufzeit löst.

Mit anderen Worten existiert kein glatter Algorithmus, der durch die Information, dass ein gegebener Graph bipartit ist, eine im Sinne der O-Notation signifikante Laufzeitreduktion erzielt. Der Beweis wird indirekt geführt, indem die Existenz eines solchen Algorithmus A' angenommen wird. Dann wird gezeigt, dass beliebige Graphen in bipartite Graphen transformiert werden können und dass die kürzesten Wege

des Graphen gegenüber dieser Transformation invariant sind. Das APSP-Problem im transformierten Graphen hat dann asymptotisch die gleiche Komplexität wie das APSP-Probelm auf Graphen, von denen die Bipartitheitseigenschaft a priori gegeben ist. Letzteres führt zum Widerspruch.

Beweis:
Für den indirekten Beweis treffen wir folgende Annahme:

Sei B ein bipartiter Graph mit n Knoten und m Kanten. Dann gibt es einen auf bipartite Graphen optimierten, glatten Algorithmus A' mit Laufzeit $f'(n, m)$, der das APSP-Problem auf B asymptotisch in signifikant kürzerer Laufzeit löst als jeder Algorithmus A mit Laufzeit $f(n, m)$, der die Bipartitheitseigenschaft von B nicht beachtet. Mit anderen Worten gilt: $f'(n, m) = o(f(n, m))$.

Im Folgenden wird eine Transformation T definiert, welche beliebige Graphen in bipartite Graphen überführt und die die folgenden Eigenschaften hat:

1. Bipartitheit: Für alle Graphen G ist $G' = T(G)$ bipartit.
2. Wege: Die kürzesten Wege in G lassen sich bestimmen durch eine Kürzeste-Wege-Suche in G', aus deren Ergebnis manche Wege gelöscht werden.
3. Größe: Die Größe des Graphen verdoppelt sich maximal in der Anzahl der Knoten und Kanten.
4. Laufzeit der Transformation: Die Laufzeit von T liegt in $O(n + m)$.
5. Laufzeit zur Extraktion der Wege: Die Laufzeit zur Extraktion der kürzesten Wege in G aus der Menge der kürzesten Wege in G' hängt linear von der Anzahl der gefundenen Wege in G' ab. Diese Extraktion wird im Folgenden mit T^- bezeichnet.

DEFINITION 3.1.11 (TRANSFORMATION T)
Sei $G = (V, E)$ ein Graph mit $|V| = n$ Knoten und $|E| = m$ Kanten. Die Transformation $T(G)$ erzeugt einen neuen Graph $G' = (V', E')$ mit $|V'| = n'$ und $|E'| = m'$. Dabei werden die folgenden beiden Fälle unterschieden:
Fall 1: $m > n$
Für jeden Knoten $v \in V$ werden in V' zwei Knoten v_i und v_o eingefügt: $V' = \{v_i, v_o | \exists v \in V\}$. Dabei erhält v_i alle eingehenden Kanten von v und v_o alle ausgehenden. Die Menge der v_i bzw. v_o wird V_i bzw. V_o genannt. Zusätzlich wird eine Kante $\langle v_i, v_o \rangle$ mit Gewicht 0 eingeführt. Somit ergibt sich: $E' = \{\langle u_o, v_i \rangle | \langle u, v \rangle \in E\} \cup \{\langle v_i, v_o \rangle | v \in V\}$. Diese Transformation ist beispielhaft in Abb. 3.3 (a) dargestellt.
Fall 2: $n \geq m$
Jede Kante $e = \langle u_1, u_2 \rangle \in E$ wird in G' durch einen neuen Knoten $v_e \in V'$ geteilt: $V' = V \cup V_e$ mit $V_e = \{v_e \mid e \in E\}$ und $E' = \{e_1 = \langle u_1, v_e \rangle, e_2 = \langle v_e, u_2 \rangle | \; e = \langle u_1, u_2 \rangle \in E$ und $v_e \in V_e\}$. Dabei erhält Kante e_1 das Gewicht 0, e_2 das Gewicht von e. Diese Transformation ist beispielhaft in Abb. 3.3 (b) dargestellt. $\Diamond$

Nach Konstruktion erfüllt T die oben geforderten Eigenschaften:

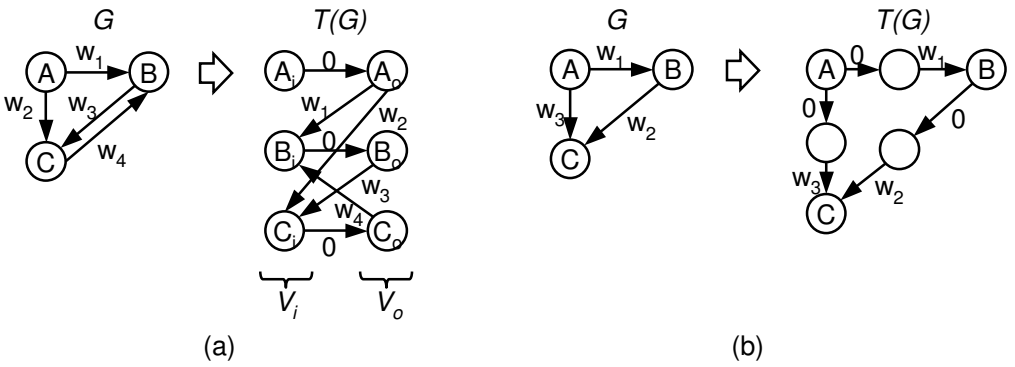


Abbildung 3.3: Transformation T angewandt auf zwei Beispielgraphen. (a) $m > n$. (b) $n \geq m$.

1. Bipartitheit: Für $m > n$ zerfällt V' in die Mengen V_i und V_o. Beide Mengen sind disjunkt und es existiert keine Kante, die zwei Knoten derselben Menge verbindet.
   Für $n \geq m$ zerfällt V' ebenfalls in zwei disjunkte Mengen: Die Menge aller in G vorhandenen Knoten V und die Menge der neu eingefügten Knoten V_e. Da alle Kanten in E durch neue Knoten unterbrochen wurden, existiert keine Kante zwischen zwei Knoten aus V. Da jeder Knoten in V_e genau eine Eingangskante und genau eine Ausgangskante zu Knoten in V hat, existiert keine Kante zwischen zwei Knoten in V_e.
2. Wege: Für $m > n$ wird jede Kante $\langle u, v\rangle$ in G auf einen Pfad $\langle\langle u_o, v_i\rangle, \langle v_i, v_o\rangle\rangle$ gleichen Gewichts in G' abgebildet. Da diese Pfade durch Kanten in G hervorgerufen werden, werden sie als Kantenpfade bezeichnet. Da die $v_o \in V_o$ nur eine einzige eingehende Kante besitzen und diese Kante von ihrem zugehörigen v_i ausgeht, sind in G' genau die v_o untereinander durch Kantenpfade verbunden, deren entsprechende Knoten in G durch Kanten verbunden sind. Da jeder Kantenpfad in G' gleiches Gewicht wie die entsprechende Kante in G hat, gilt für einen beliebigen Pfad $p = \langle v_1, \ldots v_k\rangle$ in G und seinen entsprechenden Pfad $p' = \langle v_{1_o}, \ldots v_{k_o}\rangle$ in G', dass p und p' gleiches Gewicht haben.
   Falls $n \geq m$ wird jede Kante in G auf einen Pfad gleicher Länge zwischen Knoten in $V \subset V'$ abgebildet. Außer den so entstandenen Pfaden existieren zwischen Knoten in $V \subset V'$ keine weiteren Pfade. Wiederum gilt also, dass ein beliebiger Pfad p in G und sein entsprechender Pfad p' in G' gleiches Gewicht haben.
   Wir erhalten also, dass Pfade aus G direkte Entsprechungen in G' haben und dass zwischen den betreffenden Knoten in G' keine weiteren Pfade existieren. Die kürzesten Wege in G lassen sich daher durch eine Kürzeste-Wege-Suche in G' bestimmen, aus deren Ergebnis alle Wege gelöscht werden, die nicht zwischen zwei Knoten verlaufen, die jeweils Entsprechungen in G haben.
3. Größe: Die Transformation im Fall $m > n$ erzeugt einen Graphen mit $n' = 2n$ Knoten und $m' = m + n$ Kanten. Da $m > n$ gilt $m' \leq 2m$.

Für den Fall $n \geq m$ wird ein Graph mit $n' = n + m$ Knoten und $m' = 2m$ Kanten erzeugt. Da $n \geq m$ gilt $n' \leq 2n$.
In jedem Fall kann also die Größe von G' folgendermaßen abgeschätzt werden: $n' \leq 2n$, $m' \leq 2m$.

4. Laufzeit von T: Die Laufzeit der Transformation ist in beiden Fällen linear in der Größe des Ergebnisgraphen, also für $m > n$: $O(m'+n') = O(2n+2m) = O(m)$ und für $n \geq m$: $O(m' + n') = O(2n + 2m) = O(n)$. Insgesamt liegt die Laufzeit somit in $O(m + n)$.
5. Laufzeit von T^-: In beiden Fällen wird die Liste der Ergebniswege in G' genau einmal durchlaufen, wobei jeweils Start- und Endknoten eines Weges untersucht werden.
   Für $m > n$ wird die Liste der Ergebniswege durchlaufen und nur solche Wege extrahiert, die zwei Knoten aus V_o verbinden.
   Für $n \geq m$ wird ebenfalls die Liste der Ergebniswege in G' durchlaufen. In diesem Fall werden nur Wege zwischen zwei Knoten aus $V \subset V'$ extrahiert.

Zur Berechnung der kürzesten Wege in G kann also zunächst T auf G und anschließend der auf bipartite Graphen spezialisierte Algorithmus A' auf $T(G)$ angewandt werden. Die Laufzeit von T liegt in $O(n + m)$, A' liegt in $O(f'(n', m')) = O(f'(2n, 2m))$. Die Laufzeit von T^- ist linear in der Anzahl der in G' gefundenen Pfade und liegt ebenfalls in $O(f'(n', m'))$, da die Größe des Ergebnisses eine untere Schranke für A' ist. Da A' nach Voraussetzung glatt ist, gilt $f'(2n, 2m) = O(f'(n, m))$ und $T^-(A'(T(G)))$ kann in $O(n + m) + O(f'(n, m))$ berechnet werden.

Da die triviale untere Schranke zur Berechnung von kürzesten Wegen bei $\Omega(n + m)$ liegt, gilt auch $f'(n, m) = \Omega(n + m)$, d.h. $O(n + m) = O(f'(n, m))$ und somit $O(n + m) + O(f'(n, m)) = O(f'(n, m))$.

Die Berechnung der kürzesten Wege eines beliebigen Graphen G mittels $T^-(A'(T(G)))$ liegt also genau wie die Berechnung der kürzesten Wege eines bipartiten Graphen B mittels $A'(B)$ in $O(f'(n, m))$. Dies führt zum Widerspruch mit der Annahme, dass kürzeste Wege in bipartiten Graphen asymptotisch signifikant schneller berechnet werden können als auf beliebigen Graphen. □

KOROLLAR 3.1.12 (KÜRZESTE WEGE AUF k-PARTITEN GRAPHEN)
Die Aussage aus Lemma 3.1.10 lässt sich von bipartiten auf k-partite Graphen übertragen.

Beweis: Wieder wird der Beweis indirekt geführt. Diesmal wird angenommen, es existiere ein Algorithmus, welcher die kürzesten Wege auf k-partiten Graphen für $n \geq k > 2$ schneller berechnet als der schnellste Algorithmus für bipartite Graphen. Diese Annahme führt ebenfalls zum Widerspruch, da Bipartitheit bereits k-Partitheit für $2 < k \leq n$ impliziert. □

KOROLLAR 3.1.13 (BESCHRÄNKTE KÜRZESTE WEGE)
Die Aussagen aus Lemma 3.1.10 und Korollar 3.1.12 lassen sich auf Algorithmen

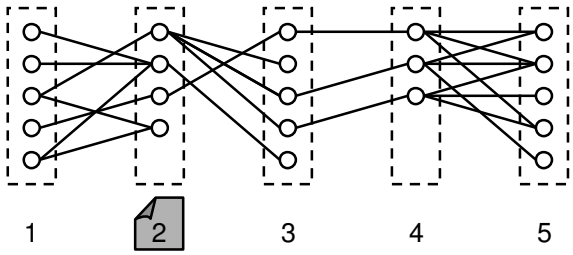


Abbildung 3.4: Ein 5-partiter Graph.

übertragen, die lediglich alle kürzesten Wege bis zu einer vorgegebenen Maximaldistanz berechnen.

Beweis:
Analog, da auch für dieses Problem die untere Schranke von $\Omega(n + m)$ gilt. □

Diskussion Die Aussage in Lemma 3.1.10 zur Kürzesten-Wege-Suche auf k-partiten Graphen zeigt, dass aus der k-Partitheit eines aus einer relationalen Datenbank generierten ContextGraphs asymptotisch kein Nutzen für die Suche nach kürzesten Wegen gezogen werden kann. Daher kommen bei der Implementierung des Systems die beiden in Abschnitt 2.7.2 vorgestellten Algorithmen zum Einsatz.

Die Aussage in Korollar 3.1.13 ist für die vorliegende Arbeit insofern von Bedeutung, als davon ausgegangen werden kann, dass für ein Dokument im ContextGraph nur dessen direkte Umgebung, d.h. dessen Kontext, relevant ist. Es müssen somit nicht alle kürzesten Wege berechnet werden, sondern lediglich die kürzesten Wege bis zu einer vorgegebenen maximalen Weglänge.

Über eine weitere Optimierung trifft Korollar 3.1.13 jedoch keine Aussage: Für die Anwendung ist es – wie oben beschrieben – ausreichend, jeweils den Kontext von *Dokumenten* zu ermitteln. Es müssen also keineswegs die kürzesten Wege ausgehend von allen Knoten berechnet werden. Für diese Optimierung könnte sich die k-Partitheit des ContextGraphs wiederum lohnen, da für große k und kleine Kontexte evtl. ganze Graphenteile a priori eliminiert und somit der Graph effektiv verkleinert werden kann. Zur Illustration betrachte man den 5-partiten Beispielgraphen in Abbildung 3.4. Es werde angenommen, dass alle Dokumente in Partition zwei liegen. Erfordert die Anwendung nun lediglich die Berechnung des Kontexts von Dokumenten mit maximaler Weglänge zwei, so können a priori alle Knoten der fünften Partition aus dem Graphen entfernt werden.

Im Gegensatz zur bisherigen Diskussion zur Optimierung der Laufzeit von Kürzeste-Wege-Algorithmen findet sich bei Goldman et al. [1998] ein Ansatz zur Optimierung des Speicherplatzes, der zum Abspeichern der Kürzeste-Wege-Information benötigt wird. Diese Art der Optimierung ist also überall dort anwendbar, wo kürzeste Wege genau einmal berechnet werden sollen und dann das Ergebnis abgespeichert und für spätere Anfragen vorgehalten werden soll. In einer indexbasier-

ten Suchmaschine würde die Berechnung also zur Indexierungszeit stattfinden, zur Anfragezeit wird dann lediglich auf die abgespeicherte Information zugegriffen.

Die Optimierung bei Goldman et al. [1998] basiert darauf, nur einen Teil der kürzesten Wege zur Indexierungszeit zu berechnen und abzuspeichern, während die restliche Berechnung zur Anfragezeit stattfindet. Dazu wird beim sogenannten *Hub-Index* der Graph in Partitionen unterteilt, welche nur durch sogenannte Hub-Knoten (etwa: Zentralknoten) verbunden sind. Zur Indexierungszeit werden dann lediglich die kürzesten Wege innerhalb einer Partition sowie zwischen Hub-Knoten berechnet. Erst zur Anfragezeit werden die kürzesten Wege zwischen Knoten unterschiedlicher Partitionen berechnet. Der Vorteil des Hub-Indexes gegenüber der Abspeicherung aller kürzesten Wege liegt in seiner geringeren Größe, der Nachteil in der aufwändigeren Berechnung zur Anfragezeit.

Für das vorliegende Szenario muss bezweifelt werden, ob der Hub-Index gewinnbringend eingesetzt werden kann, da hier nicht alle kürzesten Wege im Graph sondern nur die kürzesten Wege bis zu einer vorgegebenen Schranke berechnet werden müssen. Der durch die Aufteilung von ohnehin kurzen Pfaden eingesparte Speicherplatz dürfte somit kaum ins Gewicht fallen.

3.2 Bewertungsmaße

Das Ziel einer jeden Dokumentensuchmaschine besteht darin, zu einer Suchanfrage dem Benutzer alle relevanten Dokumente des Dokumentenstamms zu präsentieren. Für nicht-triviale Dokumentenstämme stellt sich jedoch hierbei das Problem, dass aufgrund der vom Benutzer angegebenen Suchbegriffe eine so große Anzahl an Dokumenten gefunden wird, dass eine Durchsicht aller Dokumente für den Benutzer schlichtweg unmöglich ist.

Das zentrale Werkzeug einer Suchmaschine, um diesem Dilemma zu entgehen, besteht in der Bewertung der Suchergebnisse, entsprechend ihrer vermuteten Relevanz für den Benutzer, dem sog. “Ranking”. Die Bewertung und somit die Sortierreihenfolge der Suchresultate stellt sich als umso zentraler dar, als Suchmaschinenbenutzer in der Regel nur wenige Dokumente eines Anfrageresultats beachten. Nach Jansen et al. [1998] betrachten 58% der Benutzer bei einer Anzeige von 10 Suchresultaten pro Seite nur die erste angezeigte Resultatsseite; vgl. ebenfalls die Studie von Wolfram [2000]. In der Praxis bedeutet dies, dass der Benutzer, falls unter diesen ersten Dokumenten kein relevantes zu finden war, die Suchanfrage anders formuliert oder zu anderen Suchmaschinen oder ganz anderen Methoden (z.B. Browsen, Datenbankabfragen, Expertenmeinungen, etc.) greift.

Im Rahmen der vorliegenden Arbeit stehen, wie bereits in Abschnitt 2.1.3 angesprochen, die folgenden drei Bewertungskriterien im Vordergrund:

- Relevanz eines Dokuments. Die Relevanz eines Dokuments repräsentiert die

Wahrscheinlichkeit, dass der Dokumenteninhalt den durch eine Benutzeranfrage ausgedrückten Informationsbedarf erfüllen kann, siehe Abschnitt 2.1.3.2.

- Wichtigkeit eines Dokuments. Im Unterschied zur Relevanz wird die Wichtigkeit eines Dokuments unabhängig von Benutzeranfragen bestimmt, siehe Abschnitt 2.1.3.3.

- Ähnlichkeit zweier Dokumente. Neben der populären Schlüsselwortsuche bieten viele Suchmaschinen einen Dienst, welcher zu einem gegebenen Dokument ähnliche Dokumente bestimmt, siehe Abschnitt 2.1.3.4.

Basierend auf der im ContextGraph modellierten Information wurden im Rahmen dieser Arbeit neue Bewertungsmaße für alle drei Bereiche entwickelt. Diese werden im Folgenden vorgestellt. Da kein Ansatz bekannt ist, der es ermöglicht, die Güte von Bewertungsmaßen aufgrund ihrer Definition abzuschätzen, wird im Folgenden die Diskussion jeweils an einem einfachen Beispiel vorgenommen. Diese Beispiele können per se keinen Anspruch auf Allgemeingültigkeit haben, sollen jedoch dazu dienen, die den Bewertungsmaßen zugrundeliegenden Ideen zu illustrieren. Die Implementierung und Evaluation der Maße für Relevanz und Ähnlichkeit von Dokumenten sind in Kapitel 4 und 5 beschrieben.

3.2.1 Relevanz eines Dokuments

Ein unabdingbares Werkzeug für Dokumentensuchmaschinen zur Sortierung der Suchergebnisse ist ein Maß zur Bestimmung der geschätzten Relevanz eines Dokuments bzgl. der vom Benutzer eingegebenen Suchanfrage. Bei sehr großen Ergebnismengen nimmt zwar die Bedeutung der anfrageunabhängigen Dokumentenwichtigkeit zu, es kann jedoch niemals ganz auf ein anfrageabhängiges Maß verzichtet werden, da sonst jede Suchanfrage im gleichen Ergebnis resultieren würde.

Die Schätzung der Relevanz eines Dokuments basiert üblicherweise auf einem Maß für die Übereinstimmung der vom Benutzer angegebenen Suchbegriffe mit den im Dokument enthaltenen Begriffen. Wie in Abschnitt 2.1.3.2 beschrieben, lassen sich die gängigsten Modelle zur Relevanzbewertung von Dokumenten in die Kategorien der mengentheoretischen, der algebraischen oder Vektorraum-Modelle oder der probabilistischen Modelle einteilen. Die im Folgenden vorgestellten Ideen zur Relevanzbewertung von Dokumenten basieren auf ContextGraph-Information und sind auf keines der Modelle beschränkt. Die Umsetzung der Ideen wird im Rahmen der vorliegenden Arbeit am Beispiel des Vektorraum-Modells demonstriert. Das Vektorraum-Modell wurde einerseits wegen seiner konzeptionellen Einfachheit und andererseits wegen seiner weiten Verbreitung ausgewählt. Die grundlegenden Ideen des Ansatzes könnten jedoch ebenfalls auf mengentheoretische oder probabilistische Modelle übertragen werden.

In diesem Abschnitt werden zunächst die Anforderungen zur Relevanzbewertung von Dokumenten, basierend auf ContextGraph-Information, eingeführt. Im zweiten Teil werden die Grundlagen des klassischen Vektorraum-Modells zusammen mit dem $\mathrm{tf.idf}$-Maß erläutert. Im dritten Teil wird ein neues Maß entwickelt, welches eine Erweiterung des Vektorraummodells gemäß der Anforderungen darstellt. Aus text- und kontextbasierter Bewertung wird sodann ein kombiniertes Maß abgeleitet und anschließend die Implementierung boolescher Operatoren diskutiert. Schließlich werden Anknüpfungspunkte zur Übertragung des Ansatzes auf das probabilistische Modell genannt.

3.2.1.1 Anforderungen

Die im ContextGraph modellierte Information beschreibt die Dokumente eines Unternehmens sowie ihre Kontextualisierung durch unternehmenseigene Datenbankdaten. Sowohl Dokumente als auch Kontextdaten werden dabei durch Knoten des ContextGraphs repräsentiert. Die Kanten des Graphen sind mit einem Maß für den semantischen Abstand gewichtet. Die Kontextinformation eines Dokuments kann somit klassifiziert werden nach dem semantischen Abstand zum jeweiligen Knoten.

In Abbildung 3.5 ist ein Dokumentenknoten n_d zusammen mit der Menge der Knoten seines Kontexts $\mathrm{context}_3(n_d) = \{a_1, a_2, b, c_1, c_2, d\}$ abgebildet. Für die Knoten des Kontexts gilt dabei Folgendes:

Knoten n	a_1	a_2	b	c_1	c_2	d
Beschriftung: $\mathrm{vlabel}(n)$	A	A	B	C	C	D
Abstand von n_d: $\mathrm{dist}(n_d, n)$	1	2	1	1	1	2
Nähe von n_d: $\mathrm{sim}_3(n_d, n)$	2	1	2	2	2	1

Dabei stehen A, B, C und D symbolisch für vier verschiedene Suchbegriffe. Für die unterschiedlichen Relevanzen in diesem Szenario soll gelten, dass die Relevanz des Dokuments d bzgl. des Begriffs A größer sein soll als bzgl. B. Beide Suchbegriffe kommen zwar in großer semantischer Nähe von d vor, A kommt jedoch noch zusätzlich in größerem Abstand vor. Vergleicht man Suchbegriff C mit Suchbegriff A, so sollte die Relevanz bzgl. C größer sein, da C mehrfach in nächster Nähe zu d vorkommt. Die Relevanz bzgl. Suchbegriff D schließlich sollte kleiner sein, als bzgl. B, da D wie B nur einmal im Kontext von d vorkommt, jedoch mit größerem semantischen Abstand.

Im Folgenden sei die Menge aller Suchanfragen mit $\mathbb{Q}$ bezeichnet. Eine Suchanfrage ist dabei eine Folge von Suchbegriffen. Ob die Suchbegriffe mit UND oder mit ODER verknüpft sind, soll an dieser Stelle keine Rolle spielen. Für eine Diskussion zu Booleschen Operatoren sei auf Abschnitt 3.2.1.5 verwiesen.

Die Relevanz eines Dokuments d bzgl. einer Suchanfrage $q \in \mathbb{Q}$ wird dann durch die Funktion $\mathrm{rel}(d, q)$ beschrieben. Somit ergibt sich als Anforderung für kontextba-

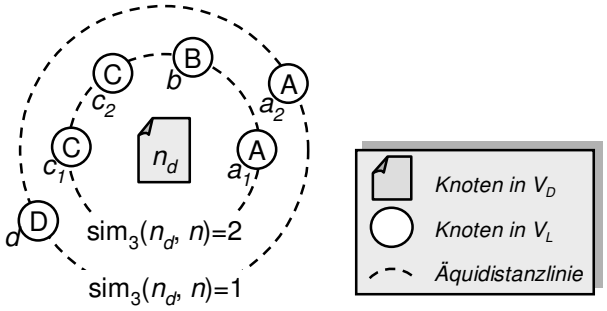


Abbildung 3.5: Dokumentenknoten n_d mit $\mathrm{context}_3(n_d)$.

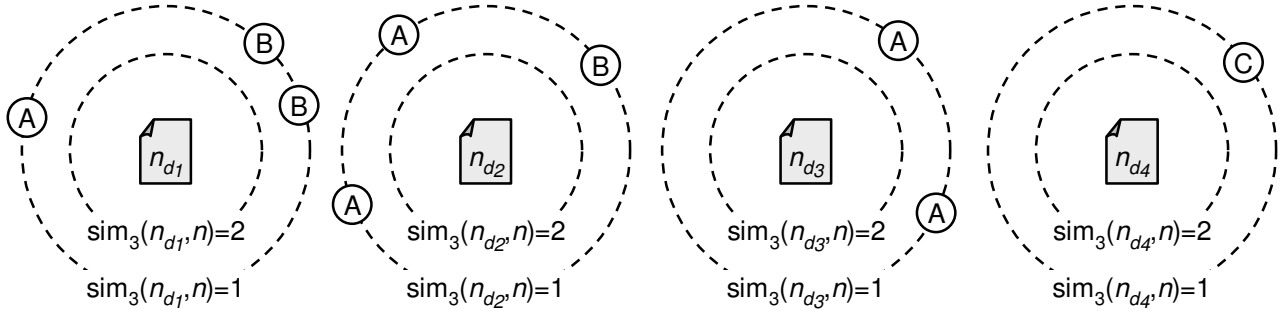


Abbildung 3.6: Dokumentenknoten $n_{d_1}, n_{d_2}, n_{d_3}, n_{d_4}$ mit jeweiligem Kontext.

sierte Relevanz:

$$\mathrm{rel}(d, \mathsf{D}) < \mathrm{rel}(d, \mathsf{B}) < \mathrm{rel}(d, \mathsf{A}) < \mathrm{rel}(d, \mathsf{C}). \tag{3.1}$$

Während im obigen Beispiel lediglich der Kontext von Dokument d für die Bewertung der Relevanz herangezogen wird, kann ebenfalls der Kontext aller im Dokumentenstamm $\mathbb{D}$ enthaltenen Dokumente miteinbezogen werden. In Abb. 3.6 ist $\mathbb{D} = \{d_1, d_2, d_3, d_4\}$. Im Kontext von d_1 und d_2 existieren Knoten mit Beschriftung A und B, im Kontext von d_3 und d_4 nur zwei Knoten mit Beschriftung A, bzw. ein Knoten mit Beschriftung C.

Gegeben sei die Suchanfrage A B und damit verbunden die Frage, welches Dokument für diese Suchanfrage am relevantesten ist. Da beide Suchbegriffe nur im Kontext von d_1 und d_2 vorkommen, sollten d_1 und d_2 relevanter sein als d_3 und d_4. Da im Kontext von d_4 keiner der Suchbegriffe auftaucht, sollte d_3 besser als d_4 bewertet werden. Nach den obigen Betrachtungen sollten d_1 und d_2 gleiche Relevanz für die Anfrage haben, da im Kontext beider Dokumente ein Suchbegriff einfach, der andere doppelt vorkommt. An dieser Stelle kann jedoch eine Abschätzung erfolgen, welcher der beiden Suchbegriffe für den Benutzer größere Signifikanz hat. Da Suchbegriff A im Kontext dreier Dokumente auftaucht, liegt die Vermutung nahe, dass die Nähe zu einem mit B beschrifteten Knoten ein besseres Alleinstellungsmerkmal ist als die Nähe zu einem mit A beschrifteten Knoten. Dem Begriff B wird daher eine größere Signifikanz zugewiesen als dem Begriff A, womit d_1 eine höhere Relevanz

als d_2 erhält, d.h.:

$$\mathrm{rel}(d_1, \mathsf{A\ B}) > \mathrm{rel}(d_2, \mathsf{A\ B}) > \mathrm{rel}(d_3, \mathsf{A\ B}) > \mathrm{rel}(d_4, \mathsf{A\ B}). \tag{3.2}$$

Die Suche nach A B auf dem Dokumentenstamm $\mathbb{D} = \{d_1, d_2, d_3, d_4\}$ kann in etwa verglichen werden mit der Suche nach "Fehlerbericht Klebenaht" auf den Dokumenten einer Qualitätssicherungsabteilung. Alle Fehlerberichte sind dort mit einem mit "Fehlerbericht" beschrifteten Knoten verbunden. Daher wird der Begriff "Fehlerbericht" als weniger signifikant eingestuft.

3.2.1.2 Das Vektorraum-Modell und tf.idf

In diesem Abschnitt werden das aus dem Information Retrieval bekannte Vektorraum-Modell und das $\mathrm{tf.idf}$-Maß kurz vorgestellt, siehe z.B. [Baeza-Yates und Ribeiro-Neto, 1999].

Das Vektorraum-Modell bestimmt in erster Linie ein Maß für die gegenseitige Relevanz oder Ähnlichkeit von Dokumenten. Im Vektorraum-Modell werden Dokumente als Vektoren repräsentiert. Zwei Dokumente gelten dann als gegenseitig relevant oder ähnlich, wenn ihre Vektorrepräsentation ähnlich ist.

Für die folgende Diskussion sei $\mathbb{D} = \{d_1, \ldots, d_D\}$ der Dokumentenstamm, d.h. die Menge der durchsuchbaren Dokumente und $\mathbb{T}^{\mathbb{D}} = \{t_1, \ldots t_{T^{\mathbb{D}}}\}$ die Menge der suchbaren Begriffe in $\mathbb{D}$. Ein Dokument $d \in \mathbb{D}$ wird als $T^{\mathbb{D}}$-dimensionaler Vektor $\vec{d}$ im Vektorraum $\vec{\mathbb{D}}$ repräsentiert; d.h. für jeden suchbaren Begriff t_i existiert genau ein Eintrag in $\vec{d}$. Diese Einträge werden im Englischen mit *term-weights* bezeichnet und im folgenden *Begriffsgewichte* genannt.

Die Ähnlichkeit von Vektoren kann auf verschiedene Arten berechnet werden. Häufig wird der Kosinus des Winkels zwischen zwei Vektoren als Maß für deren Ähnlichkeit genommen [Baeza-Yates und Ribeiro-Neto, 1999]. Seien die Dokumente $a, b \in \mathbb{D}$ durch die Vektoren $\vec{a}, \vec{b} \in \vec{\mathbb{D}}$ repräsentiert. Ihre gegenseitige Relevanz ist dann:

$$\mathrm{rel}(a, b) = \cos(\sphericalangle(\vec{a}, \vec{b})) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|} = \frac{\sum_{i=1}^{T^{\mathbb{D}}} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{T^{\mathbb{D}}} (a_i)^2} \cdot \sqrt{\sum_{i=1}^{T^{\mathbb{D}}} (b_i)^2}} \tag{3.3}$$

Die Berechnung der Begriffsgewichte ordnet jeder Kombination aus Begriff und Dokument ein Maß für die gegenseitige Relevanz zu und kann daher als Funktion $\mathrm{term\text{-}weight} : \mathbb{D} \times \mathbb{T}^{\mathbb{D}} \rightarrow \mathbb{R}^+$ aufgefasst werden. Die Berechnung der Begriffsgewichte wird im Vektorraum-Modell oft über das sog. $\mathrm{tf.idf}$-Maß realisiert. Dieses besteht aus zwei Teilen:

- Die tf-Komponente berücksichtigt die Anzahl der Vorkommen des Begriffs im Dokument. Die Abkürzung tf steht für *term frequency* (etwa: Begriffshäufig-

keit). Der tf-Wert steigt mit zunehmender Begriffshäufigkeit und wird normalisiert durch den am häufigsten im Dokument vorkommenden Begriff.

- Die idf-Komponente bezeichnet die Aussagekraft des Begriffs bezogen auf den Dokumentenstamm. Die Abkürzung idf steht für *inverse document frequency* (etwa: inverse Dokumentenhäufigkeit). Je seltener ein Begriff im Dokumentenstamm vorkommt, desto größer wird seine Aussagekraft. Der idf-Wert wird häufig heuristisch durch eine log-Funktion gedämpft um große Funktionswerte zu vermeiden. Eine theoretische Betrachtung der idf-Funktion ist in [Papineni, 2001] zu finden.

Die Berechnung von $\mathrm{tf.idf}$ für ein Dokument $d \in \mathbb{D}$ und einen Begriff $t \in \mathbb{T}^{\mathbb{D}}$ kann durch folgende Formel definiert werden

$$\mathrm{tf.idf}(d,t) = \mathrm{tf}(d,t) \cdot \mathrm{idf}(t) = \frac{\mathrm{freq}(t,d)}{\max_{l \in d}(\mathrm{freq}(l,d))} \cdot \log \frac{D}{n_t} \quad , \tag{3.4}$$

wobei $\mathrm{freq}(t,d)$ die Häufigkeit des Begriffs t in Dokument d bezeichnet und n_t die Anzahl der Dokumente, die t enthalten. Die hier gegebene Definition von $\mathrm{tf.idf}$ ist bei weitem nicht die einzige. Salton und Buckley [1987] geben eine Übersicht über eine Reihe verschiedener Abwandlungen und Erweiterungen.

Im bisher beschriebenen Vektorraum-Modell zusammen mit dem $\mathrm{tf.idf}$-Maß ist es lediglich möglich, die Ähnlichkeit von Dokumenten zu bestimmen. Um die Relevanz von Dokumenten bzgl. einer vom Benutzer eingegebenen Schlüsselwortanfrage zu berechnen, wird die Anfrage daher in ein Dokument transformiert. Auf logischer Ebene wird für eine Anfrage daher zunächst ein Dokument erstellt, welches genau aus den Begriffen der Anfrage besteht. Aus diesem Dokument wird sodann ein Vektor des Vektorraums $\vec{\mathbb{D}}$ erzeugt, dessen Einträge durch $\mathrm{tf.idf}$-Werte gegeben sind. Die Relevanzbewertung der Ergebnisdokumente wird schließlich als Dokumentenähnlichkeit bzgl. des Anfragedokuments nach Formel 3.3 berechnet.

Beispiel Die Funktionsweise des Vektorraum-Modells wird in Abb. 3.7 illustriert. In 3.7 (a) ist der Dokumentenstamm $\mathbb{D} = \{d_1, d_2, d_3, d_4\}$ abgebildet, der vier Dokumente umfasst und die vier Begriffe A, B, C und D beinhaltet. Die zugehörigen Vektoren nach Gleichung (3.4) $\vec{d_1}, \vec{d_2}, \vec{d_3}, \vec{d_4} \in \vec{\mathbb{D}}$ sind in Abb 3.7 (b) dargestellt. Das höchste Gewicht erhält Begriff D in Dokument d_4, da D in $\mathbb{D}$ nur einmal vorkommt und außerdem in d_4 am häufigsten ist. Das kleinste Gewicht erhält B in d_2, da B in $\mathbb{D}$ häufig vorkommt und nur einmal in d_2. Das häufige Vorkommen von B in $\mathbb{D}$, d.h. der niedrige idf-Wert von B, führt sogar dazu, dass B in d_3 leicht niedriger gewichtet wird als A, obwohl B doppelt so häufig auftritt wie A, d.h. $\mathrm{tf}(d_3, \mathsf{B}) = 2 \cdot \mathrm{tf}(d_3, \mathsf{A})$.

In Abb. 3.8 werden drei Anfragen q_1, q_2, q_3 an den Dokumentenstamm $\mathbb{D}$ aus Abb. 3.7 gezeigt (a) sowie deren Vektorrepräsentation $\vec{q_1}, \vec{q_2}, \vec{q_3}$ (b). Die Begriffsgewichte spiegeln dabei ihre idf-Werte wider, da in den Anfragen Begriffe nicht

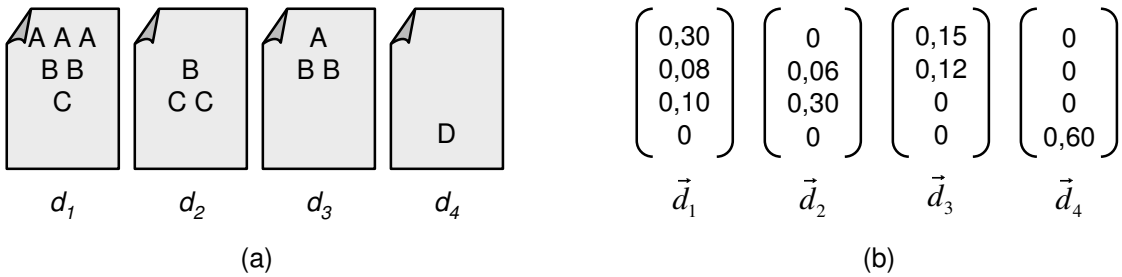


Abbildung 3.7: Beispiel zum Vektorraum-Modell mit $\mathrm{tf.idf}$. (a) Dokumente. (b) Vektoren mit $\mathrm{tf.idf}$-Gewichten nach Gleichung (3.4).

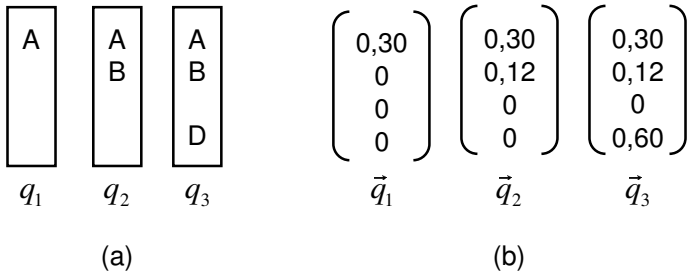


Abbildung 3.8: Beispiel zum Vektorraum-Modell mit $\mathrm{tf.idf}$. (a) Anfragen. (b) Vektoren mit $\mathrm{tf.idf}$-Gewichten nach Gleichung (3.4).

Anfrage	Rang 1	Rang 2	Rang 3	Rang 4
q_1	d_1 (0,918)	d_3 (0,769)	d_2, d_4 (0)	
q_2	d_3 (0,956)	d_1 (0,945)	d_2 (0,078)	d_4 (0)
q_3	d_4 (0,879)	d_3 (0,455)	d_1 (0,450)	d_2 (0,037)

Tabelle 3.1: Beispiel zum Vektorraum-Modell mit $\mathrm{tf.idf}$. Relevanzbewertung der Dokumente aus Abb. 3.7 bzgl. der Anfragen aus Abb. 3.8 nach Gleichung (3.3).

mehrfach vorkommen und daher für alle Begriffe $\mathrm{tf}(d,t) = 1$ gilt. Die Sortierung und Bewertung der Dokumente d_1, d_2, d_3, d_4 bzgl. der Anfragen q_1, q_2, q_3 ist in Tabelle 3.1 dargestellt. Für q_1 wird d_1 besser als d_3 bewertet, da der Begriff A häufiger vorkommt, d_2 und d_4 werden als nicht relevant eingestuft. Bezüglich der Anfragen q_2 und q_3 wird jeweils d_3 leicht besser als d_1 bewertet, da in d_1 zusätzlich der Begriff C enthalten ist, welcher nicht Teil der Anfrage ist. Für die Anfrage q_3 wird Dokument d_4 am höchsten bewertet, da es als einziges den Begriff D enthält, welcher den höchsten idf-Wert im Dokumentenstamm hat.

3.2.1.3 ContextGraph-basiertes Ranking

Gemäß der oben beschriebenen Anforderungen wird in diesem Abschnitt entsprechend der Definition von $\mathrm{tf.idf}$ ein Maß zur Berechnung der Begriffsgewichte definiert, welches die im ContextGraph modellierte Information berücksichtigt. Die

Definition dieses Maßes ist ein zentraler Aspekt der vorliegenden Arbeit und wird in den folgenden Kapiteln bzgl. Implementierung und Evaluation weiter diskutiert.

In direkter Analogie zum $\mathrm{tf.idf}$-Maß heiße die kontextbasierte Bewertungsfunktion $\mathrm{ctf.icf}$ für "Context Term Frequency" (etwa: kontextuelle Begriffshäufigkeit) und "Inverse Context Frequency" (etwa: Inverse Kontexthäufigkeit) und sei als Produkt der beiden Funktionen ctf und icf definiert.

Die zusätzliche Betrachtung der Kontexte von Dokumenten erfordert eine Erweiterung der Menge der suchbaren Begriffe. Im Folgenden bezeichnet $\mathbb{T}^{\mathbb{C}}$ die Menge der im Kontext aller Dokumente enthaltenen Begriffe.

Die ctf Funktion entspricht der tf-Funktion und bewertet die Anzahl der Knoten im Kontext eines Dokuments, die einen Suchbegriff enthalten. Während bei der tf-Funktion lediglich die Anzahl der Vorkommen bewertet werden kann, steht im kontextbasierten Fall als Zusatzinformation der semantische Abstand zwischen Dokument und Knoten zur Verfügung. Daher wird die ctf-Funktion aus der Summe aller Werte für die semantische Nähe zwischen Dokument und jeweiligem Kontextknoten berechnet.

DEFINITION 3.2.1 (CONTEXT TERM FREQUENCY (CTF))
Sei $t \in \mathbb{T}^{\mathbb{C}}$ ein Suchbegriff, $d \in \mathbb{D}$ ein Dokument. Sei $H^{t,d} = \{h_1^{t,d}, \dots\} \subseteq V_L$ die Menge der Knoten im Kontext von d, deren vlabel-Wert t enthält:

$$H^{t,d} = \{v \in \mathrm{context}(d) \mid t \in \mathrm{vlabel}(v)\}.$$

Weiterhin sei $\mathrm{contextTerms}(d)$ die Menge aller Suchbegriffe, die im Kontext von d auftreten:

$$\mathrm{contextTerms}(d) = \{t \in \mathbb{T}^{\mathbb{C}} \mid \exists n \in \mathrm{context}(d) : t \in \mathrm{vlabel}(n)\}$$

Dann wird die *Context Term Frequency* ctf von t bzgl. d als die normierte Summe aller Werte für die semantische Nähe zwischen d und den Knoten in H^t berechnet. Diese Summe wird normiert mit dem Suchbegriff aus dem Kontext von d, der eine maximale solche Summe aufweist:

$$\mathrm{ctf}(d,t) = \frac{\sum_{k=1}^{|H^{t,d}|} \mathrm{sim}(d, h_k^{t,d})}{\max_{l \in \mathrm{contextTerms}(d)} (\sum_{k=1}^{|H^{l,d}|} \mathrm{sim}(d, h_k^{l,d}))}.$$

$\Diamond$

Die icf-Funktion entspricht der idf-Funktion. Ähnlich wie tf beschränkt sich idf darauf, die *Anzahl* der Vorkommen eines Begriffs im Dokumentenstamm zu bewerten. Wiederum steht zur Definition von icf als Zusatzinformation die semantische Nähe zwischen Suchbegriff und Dokumenten zur Verfügung. Es werden daher nur solche Dokumente in die Wertung aufgenommen, die eine mindestens ebenso große semantische Nähe zu einem gegebenen Suchbegriff haben, wie das betrachtete Dokument.

DEFINITION 3.2.2 (INVERSE CONTEXT FREQUENCY (ICF))
Sei t, d und $H^{t,d}$ wie oben, $h^{t,d}_{\max} \in H^{t,d}$ derjenige Knoten aus der $H^{t,d}$, der die größte semantische Nähe zu d hat:

$$h^{t,d}_{\max} = \max_{h \in H^{t,d}} (\mathrm{sim}(d, h)).$$

Weiterhin sei $H^t \subseteq V_L$ die Menge aller Knoten, deren vlabel-Wert t enthält (die aber im Unterschied zu $H^{t,d}$ nicht aus dem Kontext eines bestimmten Dokuments entnommen sind):

$$H^t = \{v \in V_L \mid t \in \mathrm{vlabel}(v)\}.$$

Dann wird die *Inverse Context Frequency* icf von t bzgl. d berechnet aus der Anzahl der Dokumente, für die ein Knoten existiert, der t beinhaltet und mindestens die gleiche Nähe zum Dokument aufweist wie $h^{t,d}_{\max}$. Analog zu idf wird diese Summe durch die Anzahl aller Dokumente geteilt, dann der Kehrwert gebildet und schließlich durch die Anwendung der Logarithmus-Funktion gedämpft.

$$\mathrm{icf}(d, t) = \log \frac{D}{|\{\delta \in \mathbb{D} \mid \exists n \in H^t : \mathrm{sim}(\delta, n) \geq \mathrm{sim}(d, h^{t,d}_{\max})\}|}$$

$\Diamond$

Analog zu $\mathrm{tf.idf}$ wird dann $\mathrm{ctf.icf}$ folgendermaßen definiert:

DEFINITION 3.2.3 (DIE KONTEXTBASIERTE BEWERTUNGSFUNKTION CTF.ICF)
Sei $d \in \mathbb{D}$ ein Dokument, $t \in \mathbb{T}^{\mathbb{C}}$ ein Begriff und $\mathrm{ctf}(d, t)$ und $\mathrm{icf}(d, t)$ gemäß Definition 3.2.1 bzw. 3.2.2. Dann wird die kontextbasierte Bewertungsfunktion $\mathrm{ctf.icf}$ berechnet als das Produkt von ctf und icf:

$$\mathrm{ctf.icf}(d, t) = \mathrm{ctf}(d, t) \cdot \mathrm{icf}(d, t).$$

$\Diamond$

Um die Analogie von $\mathrm{ctf.icf}$ zu $\mathrm{tf.idf}$ zu veranschaulichen, sind die jeweiligen Berechnungsformeln in Tabelle 3.2 zusammengefasst.

$\mathrm{tf}(d,t) = \frac{\mathrm{freq}(t,d)}{\max_{l \in d}(\mathrm{freq}(l,d))}$	$\mathrm{ctf}(d,t) = \frac{\sum_{k=1}^{\lvert H^{t,d}\rvert} \mathrm{sim}(d, h^{t,d}_k)}{\max_{l \in \mathrm{contextTerms}(d)}(\sum_{k=1}^{\lvert H^{l,d}\rvert} \mathrm{sim}(d, h^{l,d}_k))}$
$\mathrm{idf}(t) = \log \frac{D}{n_t}$	$\mathrm{icf}(d,t) = \log \frac{D}{\lvert\{\delta \in \mathbb{D} \mid \exists n \in H^t : \mathrm{sim}(\delta, n) \geq \mathrm{sim}(d, h^{t,d}_{\max})\}\rvert}$

Tabelle 3.2: Vergleich der Funktionsdefinitionen: $\mathrm{tf}()$ vs. $\mathrm{ctf}()$ und $\mathrm{idf}()$ vs. $\mathrm{icf}()$.

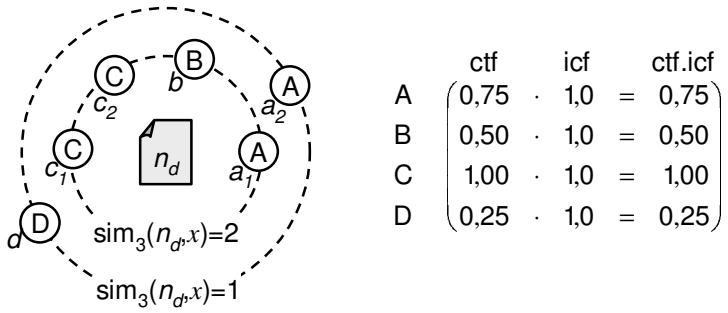


Abbildung 3.9: Beispiel zu ctf.icf nach Abb. 3.5.

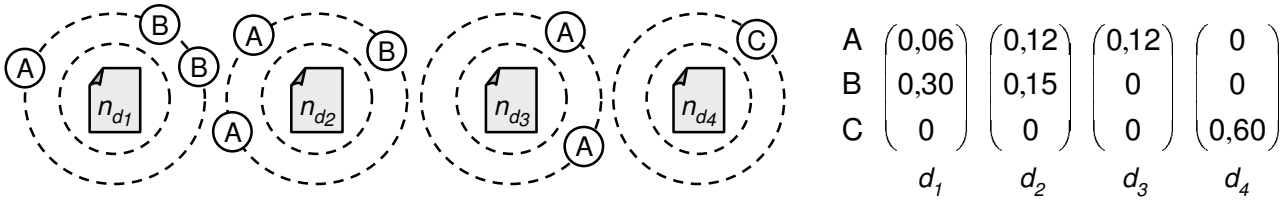


Abbildung 3.10: Beispiel zu ctf.icf nach Abb. 3.6.

Um eine Relevanzbestimmung durchführen zu können, werden im Vektorraum-Modell, wie in Abschnitt 3.2.1.2 beschrieben, Anfragen als Dokumente modelliert. Dabei wird auf logischer Ebene für jede Anfrage ein Dokument erstellt, welches genau aus den Begriffen der Anfrage besteht.

Im kontextbasierten Fall muss die Anfrage analog in den Kontext eines Dokuments transformiert werden. Dazu wird jeder Begriff der Anfrage im Kontext des Anfragedokuments mit maximalem semantischem Abstand (d.h. mit semantischer Nähe 1) eingefügt.

Beispiel Zur Illustration der Funktionsweise von ctf.icf sollen die Beispiele aus den Abbildungen 3.5 und 3.6 wieder aufgegriffen werden. Damit wird gleichzeitig überprüft, ob ctf.icf den oben formulierten Anforderungen genügt.

In Abbildung 3.9 ist das Beispiel aus Abbildung 3.5 dargestellt sowie die entsprechenden Werte für ctf.icf. Zur Berechnung der ctf.icf-Werte wird der einelementige Dokumentenstamm $\mathbb{D} = \{d\}$ angenommen. Dies führt insbesondere dazu, dass die icf-Funktion durchgehend den Wert $1,0$ annimmt. Offensichtlich ist in diesem Beispiel somit die Anforderung $\mathrm{rel}(d, \mathsf{D}) < \mathrm{rel}(d, \mathsf{B}) < \mathrm{rel}(d, \mathsf{A}) < \mathrm{rel}(d, \mathsf{C})$ aus Gleichung (3.1) erfüllt.

In Abbildung 3.10 wird das Beispiel aus Abbildung 3.6 wieder aufgegriffen sowie die jeweiligen ctf.icf-Vektoren dargestellt. Ähnlich wie die tf.idf-Funktion bewertet auch ctf.icf häufig auftretende Begriffe – im Beispiel: A – geringer als Begriffe, die seltener auftreten. Für die Anfrage A B gilt also Folgendes:

$$\begin{aligned}
\mathrm{rel}(d_1, \mathsf{A\ B}) &= 0,982 \\
\mathrm{rel}(d_2, \mathsf{A\ B}) &= 0,956 \\
\mathrm{rel}(d_3, \mathsf{A\ B}) &= 0,383 \\
\mathrm{rel}(d_4, \mathsf{A\ B}) &= 0
\end{aligned}$$

Erwartungsgemäß werden d_1 und d_2 am besten bewertet, wobei ein Knoten mit Beschriftung B etwas schwerer wiegt als ein Knoten mit Beschriftung A. Dokument d_4 ist nicht relevant für die Anfrage, da es keinen der Suchbegriffe enthält. Die Anforderung $\mathrm{rel}(d_1, \mathsf{A\ B}) > \mathrm{rel}(d_2, \mathsf{A\ B}) > \mathrm{rel}(d_3, \mathsf{A\ B}) > \mathrm{rel}(d_4, \mathsf{A\ B})$ aus Gleichung 3.2 ist somit ebenfalls erfüllt.

Disambiguierung der Anfrage Wie in Abschnitt 2.5 erläutert, spielt bei der semantischen Suche die Disambiguierung von Anfragen eine große Rolle. Ein naheliegender Ansatz besteht darin, die gegenseitige Kontextualisierung von Anfragebegriffen auszunutzen. In Abschnitt 2.5.3.3 wurde das TAP Framework besprochen, dessen Anfragedisambiguierung genau diese Eigenschaft ausnutzt. Als Beispiel wurde genannt, dass sich das Homonym "Lager" in der Anfrage "Lager Vorrat" eher auf ein Materiallager als auf ein Kugellager bezieht.

Diese Art der Anfragedisambiguierung wird von der Bewertungsfunktion $\mathrm{ctf.icf}$ inhärent durchgeführt, da sie solche Dokumente besser bewertet, die zu möglichst vielen relevanten Knoten einen möglichst kleinen Abstand haben. Wenn ein Dokument zu mehreren solcher Knoten einen kleinen Abstand hat, so impliziert dies bereits, dass die Knoten im gleichen Kontext auftreten. Im Beispiel bevorzugt die $\mathrm{ctf.icf}$-Funktion also Dokumente im Materiallagerkontext, da die Dokumente im Kugellagerkontext einen größeren Abstand zu Knoten mit dem Begriff "Vorrat" aufweisen.

3.2.1.4 Kombinierte Berechnung der Begriffsgewichte

In Abschnitt 3.2.1.2 wurde das Vektorraummodell zusammen mit $\mathrm{tf.idf}$ vorgestellt als ein Modell zur Bewertung von Suchergebnissen, welches ausschließlich auf dem textuellen Inhalt der durchsuchbaren Dokumente basiert. In Abschnitt 3.2.1.3 wurde das $\mathrm{ctf.icf}$-Maß entwickelt, welches von $\mathrm{tf.idf}$ abgeleitet wurde und sich ausschließlich auf Information aus dem Kontext der durchsuchbaren Dokumente stützt.

Selbstverständlich kann bei einer Dokumentensuchmaschine nicht darauf verzichtet werden, die Inhalte der Dokumente in die Relevanzbewertung miteinzubeziehen. Es wird daher vorgeschlagen, die Berechnung der Begriffsgewichte als eine Kombination aus $\mathrm{tf.idf}$ und $\mathrm{ctf.icf}$ zu realisieren.

DEFINITION 3.2.4 (KOMBINIERTE BEGRIFFSGEWICHTE: TF.IDF + CTF.ICF)
Sei d ein Dokument, $t \in \mathbb{T}^{\mathbb{C}} \cup \mathbb{T}^{\mathbb{D}}$ ein Suchbegriff, $\mathrm{tf.idf}(d,t)$ die Funktion zur Berechnung der Begriffsgewichte nach Gleichung (3.4) sowie $\mathrm{ctf.icf}(d,t)$ die Funktion nach Definition 3.2.3. Dann werden die kombinierten Begriffsgewichte durch die $\text{term-weight}^{(t+c)}$-Funktion wie folgt berechnet:

$$\text{term-weight}^{(t+c)}(d,t) = r_1 \cdot \mathrm{tf.idf}(d,t) + r_2 \cdot \mathrm{ctf.icf}(d,t)$$

Die Parameter $r_1, r_2 \in \mathbb{R}^+$ dienen dazu, den Einfluss der beiden Summanden zu regulieren. ◊

Die Funktion $\text{term-weight}^{(t+c)}$ zur kombinierten d.h. text- und kontextbasierten, Berechnung der Begriffsgewichte ist somit eine echte Erweiterung der $\mathrm{tf.idf}$-Funktion.

Wird diese kombinierte Funktion zur Berechnung von Begriffsgewichten in einer Suchmaschine implementiert, dann können die Parameter r_1 und r_2 je nach Realisierung systemweit vom Administrator oder anfragebezogen vom (fortgeschrittenen) Anwender definiert werden.

3.2.1.5 Boolesche Operatoren

Viele Suchmaschinen erlauben die Verknüpfung von Suchbegriffen durch die booleschen Operatoren UND, ODER, und NICHT. Diese helfen dem Anwender seinen Informationsbedarf zu spezifizieren. Eine gängige Methode zur Implementierung der Operatoren besteht darin, sie *vor* der Berechnung der Relevanzbewertung als Filter auf die Resultatmenge anzuwenden [Gospodnetić und Hatcher, 2005]. Falls Suchbegriffe nicht explizit mit Operatoren verknüpft sind, wird i. Allg. bei der Interpretation der Suchanfrage automatisch je nach Suchmaschine der UND- bzw. der ODER-Operator hinzugefügt. Man spricht dann von *impliziter* UND- bzw. ODER-Semantik.

Der UND-Operator in einer Anfrage A UND B bewirkt, dass nur solche Dokumente gefunden und bewertet werden, die sowohl A als auch B enthalten; der ODER-Operator findet zusätzlich Dokumente, die nur einen der angegebenen Begriffe enthalten.

Eine besondere Betrachtung verdient der NICHT-Operator. Mit ihm ist es möglich, die Resultatmenge einzuschränken und so besser auf den Informationsbedarf abzustimmen. Die Anfrage A UND NICHT B findet daher alle Dokumente, die zwar A jedoch nicht B enthalten. Der NICHT-Operator stellt daher ein wichtiges Werkzeug zur Präzisierung von Suchanfragen dar, welches vor allem von erfahrenen Anwendern eingesetzt wird.

Die Anwendung von UND- bzw. ODER-Operatoren auf Suchergebnisse, die mit dem oben vorgestellten kontextbasierten $\mathrm{ctf.icf}$-Maß bewertet werden, ist problemlos möglich. Lediglich die Anwendung des NICHT-Operators erfordert Beachtung, da die Verwendung des Kontexts für negierte Suchbegriffe die Resultatmenge potenziell stärker reduziert als gewünscht.

Um das Problem des NICHT-Operators zu erläutern, seien für die folgenden Beispiele die drei Suchbegriffe A, B und C gegeben, die jeweils in genau einem Knoten auftauchen, d.h. in obiger Notation: $|H^A| = |H^B| = |H^C| = 1$.

In Abb. 3.11 sind die Dokumente d_1 und d_2 dargestellt, sowie zwei Knoten, die die Begriffe A und B enthalten. Der Kontext der Dokumente wird jeweils durch einen gestrichelten Kreis angedeutet. Für einen kleinen Wert c_s von $cRange$ ist – wie in Abb. 3.11 (a) dargestellt – für die Anfrage A UND B nur Dokument d_2 relevant, da im Kontext von d_1 nur B zu finden ist. In Abb. 3.11 (b) ist die gleiche Situation, jedoch mit größerem Kontext $cRange = c_\ell$ abgebildet. Erwartungsgemäß ist nun ebenfalls d_1 für die Anfrage A UND B relevant. Das oben vorgestellte $\mathrm{ctf.icf}$-Maß trägt dabei Sorge, dass d_2 eine höhere Relevanz zugemessen wird als d_1. Für Anfragen mit UND-Operatoren verhält sich das System somit entsprechend der intuitiven Anforderung, dass größere Kontexte zu größeren Ergebnismengen führen. Wie sich leicht nachvollziehen lässt, gilt die gleiche Eigenschaft für ODER-Operatoren.

Das Verhalten des NICHT-Operators ist in Abbildung 3.12 skizziert. Abb. 3.12 (a) zeigt ein Szenario, in dem bei kleinem Kontext $cRange = c_s$ für die Anfrage A UND NICHT C Dokument d_1 relevant ist. In Abb. 3.12 (b) ist das gleiche Szenario abgebildet, jedoch wird hier durch das Vergrößern des Kontexts d_1 aus der Resultatmenge entfernt. Dieses Verhalten ist wenig intuitiv, da so evtl. Dokumente versehentlich verloren gehen. Um dieses Verhalten zu beheben, werden im Folgenden zwei Möglichkeiten diskutiert: Negative Begriffsgewichte und adaptive Kontextgröße.

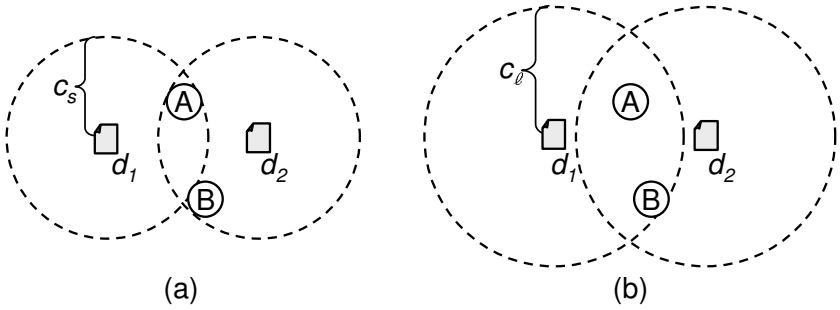


Abbildung 3.11: Beispiel zur UND-Verknüpfung. Anfrage A UND B mit Dokumenten d_1 und d_2. (a) Kleiner Kontext: $cRange = c_s$. Nur Dokument d_2 ist relevant. (b) Großer Kontext: $cRange = c_\ell$. Beide Dokumente d_1 und d_2 sind relevant.

Negative Begriffsgewichte Ein naheliegender Ansatz zum Umgang mit der Problematik negierter Suchbegriffe liegt darin, das Vorhandensein eines Begriffs im Kontext eines Dokuments durch die Negierung der Begriffsbewertung im entsprechenden Eintrag des Dokumentenvektors zu reflektieren. Damit würden solche Dokumente mehr "bestraft", die näher an Knoten platziert sind, die negierte Suchbegriffe enthalten. Gleichzeitig werden Dokumente, in deren Kontext der fragliche Suchbegriff nur in größerem Abstand vorkommt, nur geringfügig abgewertet.

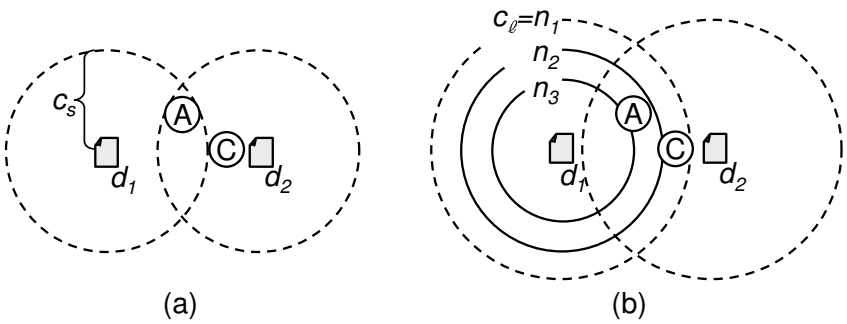


Abbildung 3.12: Beispiel zur NICHT-Verknüpfung. Anfrage A UND NICHT C mit Dokumenten d_1 und d_2. (a) Kleiner Kontext: $cRange = c_s$. Dokument d_1 ist relevant. (b) Großer Kontext: $cRange = c_\ell$. Bei adaptiver Kontextgröße beeinflussen verschiedene Werte n_1, n_2, n_3 für $nRange$ die Größe der Resultatmenge.

Dieser Ansatz hat den Vorteil, dass das $\mathrm{ctf.icf}$-Maß intuitiv auf die Behandlung negierter Suchbegriffe erweitert werden kann. Er hat jedoch den Nachteil, dass die Filterung der Dokumente, basierend auf den booleschen Verknüpfungen der Anfragebegriffe, nicht länger *vor* der Bewertung der Suchresultate vorgenommen werden kann. Zusätzlich muss im Unterschied zum normalen Vektorraum-Modell bei der Berechnung der Relevanz zwischen negierten und nicht-negierten Suchbegriffen unterschieden werden. Diese Unterscheidung ist im normalen Vektorraum-Modell, welches nur positive Begriffsgewichte vorsieht, nicht möglich.

Adaptive Kontextgröße Neben der Einführung negativer Begriffsgewichte, besteht die Möglichkeit, für negierte Suchbegriffe die Kontextgröße anzupassen. Um dies zu realisieren, wird der Parameter $nRange$ eingeführt, welcher den Einfluss negierter Suchbegriffe beschreibt. In Abbildung 3.12 (b) sind drei verschiedene Werte n_1, n_2 und n_3 für $nRange$ dargestellt. Je nach Parametrisierung von $nRange$ wird daher der negierte Begriff C als relevant für d_1 eingestuft oder nicht. Für $nRange = c_\ell = n_1$ wird C beachtet und daher d_1 aus der Resultatmenge entfernt, für $nRange = n_2$ oder $nRange = n_3$ verbleibt d_1 jedoch im Ergebnis.

Die Definition des $nRange$-Parameters kann dabei entweder durch den (fortgeschrittenen) Anwender oder durch folgenden einfachen Algorithmus erfolgen:

```
1  set nRange = cRange;
2  resultSet = processQuery();
3  while resultSet is empty AND nRange > 0 {
4    set nRange = nRange - 1;
5    resultSet = processQuery();
6  }
```

Dieser Algorithmus verfolgt das Ziel, möglichst nur solche Dokumente in der Ergebnismenge zu belassen, die möglichst weit von Knoten entfernt sind, die einen negierten Suchbegriff enthalten. Zu diesem Zweck beschneidet der Algorithmus die

Ergebnismenge zunächst maximal indem $nRange$=$cRange$ gesetzt wird. Dies resultiert in einer Ergebnismenge, in der nur solche Dokumente enthalten sind, in deren Kontext der negierte Begriff nicht auftaucht. Ist diese Ergebnismenge leer, so wird schrittweise der Wert für $nRange$ reduziert, solange bis Dokumente gefunden werden oder $nRange = 0$ ist. Im letzteren Fall hat der negierte Begriff keinen kontextbasierten Einfluss mehr auf die Resultatmenge. Wird also z.B. nach A UND NICHT C gesucht und alle Dokumente enthalten C in ihrem nächsten Kontext, d.h. die Ergebnismengen aller Suchen mit $nRange \geq 1$ sind leer, dann werden in der letzten Iteration des Algorithmus solche Dokumente gefunden, die zwar C in ihrem Kontext enthalten, nicht jedoch in ihrem Text.

Dieser Ansatz der adaptiven Kontextgröße hat den Nachteil, dass evtl. mehrere Anfragen an das System abgesetzt werden müssen. Der Vorteil liegt darin, dass er im Rahmen des Vektorraummodells realisierbar ist.

3.2.1.6 Anwendung auf das Probabilistische Modell

In diesem Kapitel wurde bisher beschrieben, wie die Anforderungen an ein kontextbasiertes Bewertungsmaß als Erweiterung eines $\mathrm{tf.idf}$-basierten Vektorraummodells realisiert werden können. Die eigentliche Bewertung der Relevanz eines Dokuments bzgl. einer Anfrage wird dabei auf ein Distanzmaß zwischen Vektoren zurückgeführt, die das Dokument incl. Kontext bzw. die Anfrage repräsentieren.

Die Erweiterung von $\mathrm{tf.idf}$ zu $\mathrm{ctf.icf}$ ist dabei jedoch nicht auf das Vektorraummodell beschränkt. Im probabilistischen Modell wird die Bewertung der Suchergebnisse aus der Wahrscheinlichkeit berechnet, mit der ein Dokument für eine Anfrage relevant ist [Spärck Jones et al., 2000]. Um diese Relevanz abzuschätzen, wurden verschiedene Maße vorgeschlagen, darunter ebenfalls ein tf-ähnliches Maß, welches die Häufigkeit von Begriffen in Dokumenten berücksichtigt und im Bereich der probabilistischen Modelle oftmals mit *Eliteness* bezeichnet wird. Neben der Eliteness existiert ein Maß für die Signifikanz von Begriffen, welches idf entspricht. Historisch betrachtet, entstammt das idf-Maß sogar dem Bereich der probabilistischen Modelle, wo es unter dem Namen *Collection Frequency Weight* (CFW) von Spärck Jones [1972] vorgeschlagen wurde.

Die dieser Arbeit zugrundeliegende Annahme, dass Begriffe aus dem Kontext eines Dokuments für dessen Inhalt signifikant sind, kann somit in einem probabilistischen Modell in Analogie zur Erweiterung der $\mathrm{tf.idf}$-Funktion in der Formel zur Berechnung der Relevanz-Wahrscheinlichkeit reflektiert werden. In einem ähnlichen Ansatz schlagen Mihajlović et al. [2005] vor, verschiedene XML-Retrieval-Modelle unter Zuhilfenahme einer Algebra zu abstrahieren. Eine detaillierte Ausführung dieser Diskussion würde jedoch den Rahmen der vorliegenden Arbeit sprengen.

3.2.2 Wichtigkeit eines Dokuments

Im vorigen Abschnitt wurde ein kontextbasiertes Maß zur Bewertung der Relevanz von Dokumenten bezüglich Suchanfragen berechnet. Eine wichtige Komponente bei der Bewertung von Dokumenten stellt jedoch eine anfrageunabhängige Bewertung der Wichtigkeit von Dokumenten dar. So ist der Erfolg der Suchmaschine Google[1] zumindest in Teilen auf die dort implementierte Berechnung der Wichtigkeit von Dokumenten, den sog. PageRank, zurückzuführen. Die Berechnung des PageRanks verwendet dabei die durch Hyperlinks induzierte Struktur des Web. Die Wichtigkeit einer Seite ergibt sich dabei aus der Anzahl und der Wichtigkeit der Seiten, die auf sie per Hyperlink verweisen. Weitere Arten zur Bewertung der Wichtigkeit sind in Abschnitt 2.1.3.3 aufgezählt.

In diesem Abschnitt wird erläutert, wie die Bewertung der Wichtigkeit von Dokumenten durch Informationen des ContextGraphs unterstützt werden kann. Dazu werden zuerst die Definitionen von PageRank sowie einer abgeleiteten Form zusammengefasst und sodann auf den ContextGraph übertragen.

3.2.2.1 PageRank

Der in Google implementierte PageRank [Brin und Page, 1998] orientiert sich am sog. Random-Surfer-Modell. Dabei wird der PageRank einer Seite als Wahrscheinlichkeit verstanden, dass ein beliebiger Benutzer (als "Surfer" im Web) diese Seite abruft. Das Verhalten des Benutzers wird folgendermaßen modelliert: Der Surf-Vorgang beginnt auf einer beliebigen Seite des Web, die vom Surfer per Suchmaschine oder Lesezeichen aufgesucht wird. Von dort aus verfolgt der Surfer einen von dort ausgehenden Link und ruft somit eine benachbarte Seite ab. Dieser Schritt wird solange wiederholt, bis der Surfer "gelangweilt" ist und wiederum per Suchmaschine oder Lesezeichen auf eine beliebige andere Seite springt. Die Wahrscheinlichkeit, dass der Surfer von einer Seite aus einen bestimmten Link verfolgt, wird dabei umso geringer, je größer der Ausgangsgrad OutDeg der Seite ist.

Der PageRank einer Seite s, berechnet sich nach folgender Formel

$$\text{PageRank}(s) = (1-d) + d \cdot \left(\frac{\text{PageRank}(t_1)}{\text{OutDeg}(t_1)} + \dots \frac{\text{PageRank}(t_n)}{\text{OutDeg}(t_n)} \right) ,$$

wobei

- d die Wahrscheinlichkeit ist, dass der Surfer einem Link einer Seite folgt,
- $(1-d)$ die Wahrscheinlichkeit ist, dass eine beliebige andere Seite angesprungen wird,
- $t_1, \dots t_n$ die Seiten bezeichnen, die einen Link auf s enthalten und

[1] `http://www.google.com/` [05.04.2006]

- $\text{OutDeg}(t_i)$ die Anzahl der Links auf der Seite t_i ist.

Offensichtlich erhalten mit dieser Berechnungsvorschrift solche Seiten einen hohen Wert, die von vielen Seiten verlinkt werden, die ihrerseits einen hohen Wert haben und gleichzeitig wenige andere Seiten verlinken. Die Berechnung des PageRanks wird in der Regel iterativ durchgeführt, wobei die Berechnung angehalten wird, sobald sich in einer Iteration die Bewertung aller Seiten um weniger als einen vorgegebenen Schwellwert ändert.

3.2.2.2 ObjectRank

Der ObjectRank-Ansatz nach Balmin et al. [2004] erweitert den PageRank zur Verwendung auf Datenbanken. Anstelle der Hyperlink-Struktur, welche dem PageRank-Algorithmus zugrunde liegt, arbeitet ObjectRank auf einem gerichteten Datenbankgraphen, dessen Kanten mit einem Wert für ihren sog. "Authority-Transfer" (in diesem Zusammenhang etwa: Weitergabe von Fachwissen) versehen sind. Der Authority-Transfer-Wert gibt den inhaltlichen Zusammenhang zweier Knoten des Datenbankgraphen an. Das Random-Surfer-Modell bei PageRank wird bei ObjectRank in ein Random-Querier-Modell überführt, welches einen Benutzer simuliert, der im Datenbankgraphen navigiert, bzw. direkt mittels SQL-Anfragen auf der zugrundeliegenden Datenbank arbeitet.

Für die Zuweisung des Authority-Transfer-Werts werden die Kanten des Datenbankgraphen zu Typen zusammengefasst, wobei solche Kanten denselben Typ erhalten, die die gleiche Fremdschlüsselbeziehung zwischen zwei Tabellen repräsentieren. Alle Kanten eines Typs erhalten denselben Authority-Transfer-Wert.

Die Berechnung des ObjectRank-Werts geschieht in Analogie zur Berechnung des PageRank-Werts mit folgenden Erweiterungen:

- Authority-Transfer. Neben den ObjectRank-Werten aller auf ein Objekt verweisenden Objekte wird zusätzlich der Authority-Transfer-Wert der verbindenden Kanten mitberücksichtigt.

- Kantenausgangsgrad. Die Berechnung des Kantenausgangsgrads eines Objekts berücksichtigt nicht alle Kanten, die ein Objekt verlassen, sondern lediglich die Kanten gleichen Typs.

Der ObjectRank eines Knoten im Datenbankgraphen s berechnet sich demnach analog zum PageRank mittels

$$\text{ObjectRank}(s) = (1-d) + d \cdot \left(\sum_{i=1\ldots n} \frac{\text{ObjectRank}(t_i) \cdot \text{AuthTransf}(t_i, s)}{\text{OutDeg}^*(t_i, s)} \right)$$

wobei:

- d die Wahrscheinlichkeit ist, dass der Random-Surfer von einem Knoten aus einer beliebigen Ausgangskante folgt,
- $(1 - d)$ die Wahrscheinlichkeit ist, dass ein beliebiger anderer Knoten angesprungen wird,
- AuthTransf(t_i, s) der Authority-Transfer-Wert der Kante $\langle t_i, s\rangle$ ist,
- $t_1, \ldots t_n$ die Knoten bezeichnen, die eine Kante zu s enthalten und
- OutDeg$^*(t_i, s)$ die Anzahl der Ausgangskanten von t_i ist, die den selben Typ wie die Kante $\langle t_i, s\rangle$ haben.

3.2.2.3 Anwendung auf den ContextGraph

Im Bereich der Wichtigkeit von Dokumenten lässt sich die Berechnung des ObjectRank-Werts direkt auf den ContextGraph übertragen, da auch im ContextGraph Kantengewichte existieren, aus denen sich ein Authority-Transfer-Wert errechnen lässt. So leistet die in Definition 3.1.6 eingeführte sim-Funktion genau das Gewünschte. Für die kontextbasierte Berechnung der anfrageunabhängigen Wichtigkeit von Dokumenten kann somit folgende Definition des ObjectRanks verwendet werden:

$$\text{ObjectRank}(s) = (1 - d) + d \cdot \left(\sum_{i=1\ldots n} \frac{\text{ObjectRank}(t_i) \cdot \mathrm{sim}(t_i, s)}{\text{OutDeg}^*(t_i, s)} \right).$$

Da das Konzept zur kontextbasierten Berechnung der Dokumentenwichtigkeit mittels ObjectRank nicht im Rahmen dieser Arbeit entstand, sondern lediglich auf das ContextGraph-Szenario übertragen wurde, wird auf eine Evaluation verzichtet, für eine solche siehe Balmin et al. [2004]. Zwar beinhaltet sowohl die in Abschnitt 4.1 vorgestellte Architektur als auch der implementierte Prototyp eine Komponente zur ContextGraph-basierten Berechnung des ObjectRanks, eine detaillierte Analyse der Auswirkung dieser Komponente auf das Laufzeitverhalten und auf die Qualität der Suchergebnisse wurde jedoch nicht durchgeführt.

3.2.3 Suche nach ähnlichen Dokumenten

Neben der schlüsselwortbasierten Dokumentensuche ist die Suche nach ähnlichen Dokumenten, wie bereits in Abschnitt 2.1.3.4 angesprochen, ein wichtiges Werkzeug zur Erschließung von Dokumentenbeständen und wird daher von vielen kommerziellen Suchmaschinen angeboten. In diesem Abschnitt wird ein Maß zur Bestimmung der Ähnlichkeit von Dokumenten vorgestellt, welches auf den im ContextGraph gespeicherten Informationen beruht. Analog zur ContextGraph-basierten Schlüsselwortsuche sollen auch hier die textbasierten Maße nicht ersetzt, sondern um einen kontextbasierten Anteil erweitert werden.

Wie in Abschnitt 2.1.3.4 vorgestellt, haben die heute verwendeten, kontextbasierten Ähnlichkeitsmaße ihren Ursprung in der Bibliometrie. Zwei Dokumente werden als ähnlich bewertet, wenn viele Dokumente auf beide verweisen (Kozitation), bzw. beide Dokumente auf gleiche Dokumente verweisen (bibliographische Kopplung). Vor dem Hintergrund des ContextGraphs lässt sich dieser Ansatz dahingehend verallgemeinern, dass zwei Dokumente in dem Maße als ähnlich bewertet werden, wie sich ihre Kontexte überlappen.

Analog zur ContextGraph-basierten Schlüsselwortsuche wird auch die Suche nach ähnlichen Dokumenten anhand des Vektorraummodells illustriert, d.h. die Ähnlichkeit zwischen Dokumenten wird auf die Ähnlichkeit zwischen Vektoren abgebildet. Im Unterschied zur Schlüsselwortsuche, bei der sich die Dimensionalität des Vektorraums nach der Anzahl der suchbaren Begriffe $|\mathbb{T}^{\mathbb{C}} \cup \mathbb{T}^{\mathbb{D}}|$ richtet, wird die Suche nach ähnlichen Dokumenten in einem Vektorraum durchgeführt, dessen Dimension der Anzahl der Knoten $|V|$ im ContextGraph entspricht. Für die Berechnung der Begriffsgewichte wird dann die im Folgenden definierte Funktion $\mathrm{nr.inf}$ angewandt, die sich analog zu $\mathrm{ctf.icf}$ an der Definition von $\mathrm{tf.idf}$ orientiert und somit aus zwei Teilen besteht.

Die nr-Komponente entspricht der tf-Komponente und beschreibt die Relevanz eines Knoten für ein Dokument. Die Abkürzung nr steht dabei für "node relevance" (etwa: Relevanz eines Knoten). Da jeder Knoten im ContextGraph genau einmal auftaucht, wird hier jedoch nicht die Häufigkeit des Auftauchens des Knoten, sondern lediglich seine semantische Nähe zum Dokument berücksichtigt. Die nr-Komponente sorgt somit dafür, dass zwei Dokumente dann als ähnlich betrachtet werden, wenn beide eine große Nähe zu einem gemeinsamen Knoten aufweisen. Analog zu tf und ctf wird auch nr normiert mit dem Abstandswert des dem Dokument am nächsten liegenden Knoten.

Die inf-Komponente bewertet entsprechend zur idf-Komponente die Signifikanz eines Knoten. Die Abkürzung inf steht für "inverse node frequency" (etwa: inverse Knotenhäufigkeit). Durch inf werden solche Knoten als signifikant eingestuft, die in den Kontexten nur weniger Dokumente auftauchen. Auf diese Weise wird verhindert, dass Knoten, die in vielen Kontexten enthalten sind, einen zu großen Einfluss auf die Berechnung der Dokumentenähnlichkeit ausüben. $\mathrm{nr.inf}$ lässt sich somit folgendermaßen definieren:

DEFINITION 3.2.5 (DIE KONTEXTBASIERTE BEWERTUNGSFUNKTION NR.INF)
Sei $d \in \mathbb{D}$ ein Dokument, $n \in V$ ein Knoten und sim wie in Definition 3.1.6. Dann wird $\mathrm{nr.inf}$ wie folgt definiert:

$$\begin{aligned}\mathrm{nr.inf}(d,n) &= \mathrm{nr}(d,n) \cdot \mathrm{inf}(d,n) \\ &= \frac{\mathrm{sim}(d,n)}{\max_{m \in V} \mathrm{sim}(d,m)} \cdot \log \frac{D}{|\{\delta \in \mathbb{D} \mid \mathrm{sim}(\delta,n) \geq \mathrm{sim}(d,n)\}|}\end{aligned}$$

$\diamondsuit$

Wiederum in Analogie zur kombinierten Berechnung der Begriffsgewichte mit $\mathrm{tf.idf}$ und $\mathrm{ctf.icf}$ soll auch bei der Suche nach ähnlichen Dokumenten nicht nur der Kontext, sondern ebenfalls der Inhalt der Dokumente berücksichtigt werden. Die Dimension des zugrundeliegenden Vektorraums ergibt sich als die Summe der Anzahl der Begriffe im Dokumentenstamm $T^{\mathbb{D}}$ und der Anzahl der Knoten im ContextGraph $|V|$. Jeder Vektor dieses Raums habe die Form $v = \langle v_1, \ldots, v_{|T^{\mathbb{D}}|}, v_{|T^{\mathbb{D}}|+1}, \ldots, v_{|T^{\mathbb{D}}|+|V|} \rangle$, d.h. die Einträge für Begriffe werden gefolgt von den Einträgen für Knoten.

DEFINITION 3.2.6 (KOMBINIERTE BEGRIFFSGEWICHTE: TF.IDF + NR.INF)
Sei d ein Dokument, $\mathrm{term} : \mathbb{N} \to \mathbb{T}$, eine Funktion, die jedem $1 \leq i \leq |T^{\mathbb{D}}|$ genau einen Begriff in $\mathbb{T}^{\mathbb{D}}$ zuweist und $\mathrm{node} : \mathbb{N} \to V$ eine Funktion, die jedem $1 \leq i \leq |V|$ genau einen Knoten in V zuweist. Dann gilt für den i-ten Eintrag im Begriffsgewichtsvektor von d:

$$\text{term-weight}^{(t+n)}(d, i) = \begin{cases} r_1 \cdot \mathrm{tf.idf}(d, \mathrm{term}(i)) & i \leq |T^{\mathbb{D}}| \\ r_2 \cdot \mathrm{nr.inf}(d, \mathrm{node}(i - |T^{\mathbb{D}}|)) & \text{sonst.} \end{cases}$$

Wieder dienen die Parameter $r_1, r_2 \in \mathbb{R}^+$ dazu, den Einfluss von Text und Kontext zu regulieren. $\diamondsuit$

3.2.3.1 Die Bewertungsmaße nr.inf und ctf.icf im Vergleich

Die Definition von $\mathrm{nr.inf}$ ist ähnlich, jedoch nicht gleich zu $\mathrm{ctf.icf}$. Zur besseren Übersicht sind die Formeln in Tabelle 3.3 zusammengefasst. Nur zu Referenzzwecken wurden ebenfalls die Definitionen von tf und idf in die Tabelle aufgenommen.

Es fällt auf, dass nr im Vergleich zu ctf weder in Zähler noch Nenner eine Summation erfordert. Der Grund hierfür liegt darin, dass bei nr keine Begriffe sondern ausschließlich Knoten in die Berechnung einfließen und jeder Knoten genau einmal im ContextGraph enthalten ist.

Weiterhin ist anzumerken, dass $\mathrm{ctf.icf}$ und $\mathrm{nr.inf}$ auf Vektorräumen unterschiedlicher Dimensionalität operieren. Während $\mathrm{ctf.icf}$ wie $\mathrm{tf.idf}$ auf einem Vektorraum definiert ist, dessen Dimensionalität durch die Anzahl der suchbaren Begriffe bestimmt wird, ist $\mathrm{nr.inf}$ auf einem Raum definiert, dessen Dimension durch die Anzahl der Knoten des ContextGraphs bestimmt wird. Die Intention dieser Festlegung liegt darin, dass die Ähnlichkeit zweier Dokumente nicht über Schlüsselworte, sondern über ihre tatsächliche Verbindung im ContextGraph bestimmt werden kann. Dieser Unterschied wird im folgenden Beispiel illustriert.

Beispiel Abbildung 3.13 (a) zeigt die drei Dokumente d_1, d_2 und d_3 sowie einen Ausschnitt aus ihrem jeweiligen Kontext. Die Dokumente d_1 und d_2 sind vom selben Autor (im Bsp.: Max Blau) verfasst, daher überlappen sich ihre Kontexte zumindest

$\mathrm{tf}(d,t) = \frac{\mathrm{freq}(t,d)}{\max_{l \in d}(\mathrm{freq}(l,d))}$	$\mathrm{ctf}(d,t) = \frac{\sum_{k=1}^{\lvert H^{t,d}\rvert} \mathrm{sim}(d,h_k^{t,d})}{\max_{l \in \mathrm{contextTerms}(d)}(\sum_{k=1}^{\lvert H^{l,d}\rvert} \mathrm{sim}(d,h_k^{l,d}))}$ $\mathrm{nr}(d,n) = \frac{\mathrm{sim}(d,n)}{\max_{m \in V} \mathrm{sim}(d,m)}$
$\mathrm{idf}(t) = \log \frac{D}{n_t}$	$\mathrm{icf}(d,t) = \log \frac{D}{\lvert\{\delta \in \mathbb{D} \mid \exists n \in H^t : \mathrm{sim}(\delta,n) \geq \mathrm{sim}(d,h_{\max})\}\rvert}$ $\mathrm{inf}(d,n) = \log \frac{D}{\lvert\{\delta \in \mathbb{D} \mid \mathrm{sim}(\delta,n) \geq \mathrm{sim}(d,n)\}\rvert}$

Tabelle 3.3: Vergleich der Funktionsdefinitionen: $\mathrm{tf}()$ vs. $\mathrm{ctf}()$ und $\mathrm{nr}()$; $\mathrm{idf}()$ vs. $\mathrm{icf}()$ und $\mathrm{inf}()$.

im Knoten, der mit Max Blau beschriftet ist. Dokument d_3 hingegen ist mit diesem Knoten nicht verknüpft, es wurde von Autor Max Braun erstellt.

In Abb. 3.13 (b) ist die Situation im $|T^{\mathbb{C}}|$-dimensionalen Vektorraum wie für die Berechnung der Begriffsgewichte mit $\mathrm{ctf.icf}$ dargestellt. Alle drei Dokumente sind sich ähnlich, da im Kontext jedes Dokuments der Begriff Max enthalten ist. Zwar würde bei einer Suchanfrage nach ähnlichen Dokumenten zu d_1 Dokument d_2 als ähnlicher bewertet werden, da im Kontext von d_2 sowohl Max als auch Blau enthalten ist. Das Dokument d_3 würde jedoch ebenfalls in die Treffermenge mitaufgenommen, obwohl sich aus dem ContextGraph ein Zusammenhang mit d_1 nicht ergibt.

Die Situation im $|V|$-dimensionalen Vektorraum wie bei $\mathrm{nr.inf}$ wird in Abb. 3.13 (c) dargestellt. Bei einer Suche nach ähnlichen Dokumenten zu d_1 wird lediglich d_2 gefunden, da die in den Vektoren mittels des $\mathrm{nr.inf}$-Maßes kodierte Information direkt die Struktur des ContextGraphs widerspiegelt.

Diskussion Zusammenfassend lässt sich sagen, dass zur Bewertung der Ähnlichkeit von Dokumenten der $|\mathbb{T}^{\mathbb{C}}|$-dimensionale Vektorraum mit $\mathrm{ctf.icf}$ und der $|V|$-dimensionale Vektorraum mit $\mathrm{nr.inf}$ alternative Ansätze darstellen, die beide erfolgreich sein können. Während im $|\mathbb{T}^{\mathbb{C}}|$-dimensionalen Vektorraum Dokumente über das Auftreten von Begriffen verknüpft werden, geschieht dies im $|V|$-dimensionalen Raum ausschließlich entlang von im ContextGraph modellierter Verbindungen.

Da in der Regel im ContextGraph die Anzahl der Knoten kleiner ist als die Anzahl der durchsuchbaren Begriffe, kann die Bewertung der Dokumentenähnlichkeit mittels $\mathrm{nr.inf}$ mit der Technik des Latent Semantic Indexing (LSI), etwa: "Einordnung verborgener Bedeutungen", verglichen werden [Deerwester et al., 1990]. Beim LSI werden verschiedene, synonym oder beinahe synonym verwendete Begriffe auf

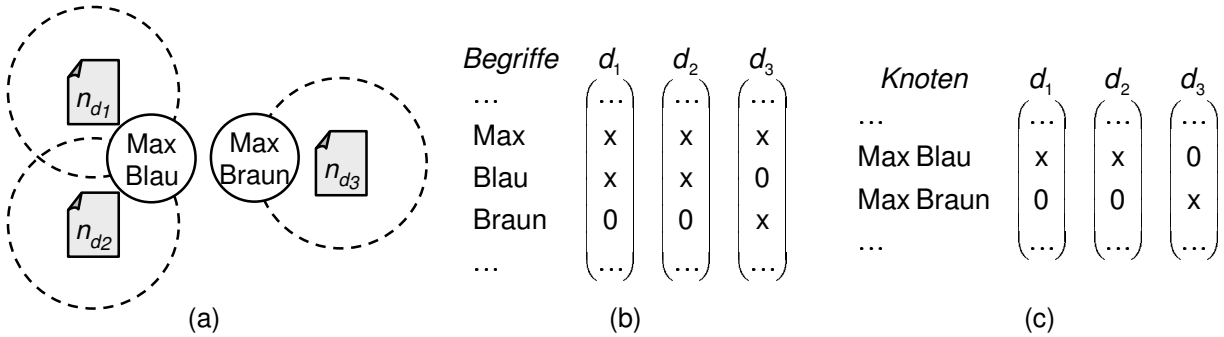


Abbildung 3.13: (a) Dokumente d_1, d_2 und d_3 in ihrem Kontext. (b) Vektoren im $|\mathbb{T}^{\mathbb{C}}|$-dimensionalen Vektorraum zur Berechnung von $\mathrm{ctf.icf}$. (c) Vektoren im $|V|$-dimensionalen Vektorraum zur Berechnung von $\mathrm{nr.inf}$. Der Platzhalter x bezeichnet Vektoreinträge > 0.

ein einziges Konzept projiziert. Diese Projektion führt zu einer Verminderung der Dimensionalität des Vektorraums, die dann nicht mehr durch die Anzahl der Begriffe, sondern durch die Anzahl der Konzepte bestimmt wird. Dies hat den Vorteil, dass die Suche nicht auf dem Raum der Begriffe, sondern auf dem sog. semantischen Raum der Konzepte durchgeführt werden kann.

Die Projektion des Begriffsraums auf den Raum der Konzepte wird bei LSI aufwändig mittels einer Singulärwertzerlegung einer Matrix errechnet, die sich aus allen Dokumentvektoren zusammensetzt. Im Gegensatz dazu erfordert die Reduktion auf $|V|$ Dimensionen für die Anwendung von $\mathrm{nr.inf}$ keinerlei Berechnung, da die Zugehörigkeit von Begriffen zu Knoten im ContextGraph bereits explizit modelliert ist.

3.3 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Konzepte für eine kontextbasierte Intranetsuchmaschine entwickelt.

Der ContextGraph dient als Abstraktion von Unternehmensdatenbanken und als Basis zur Berechnung des Kontexts von Dokumenten. Bei der Einbettung relationaler Datenbanken in den ContextGraph werden Relationen, Tupel und Attributwerte auf Knoten abgebildet, Fremdschlüsselbeziehungen zwischen Tupeln sowie die Zugehörigkeit von Tupeln zu Tabellen und von Attributwerten zu Tupeln werden durch Kanten repräsentiert.

Die Kanten des ContextGraph werden mit einem Maß für den semantischen Abstand zwischen den jeweiligen Knoten gewichtet. Die Additivität der Kantengewichte legt dabei nahe, dass der semantische Abstand asymmetrisch festgelegt wird. Aus der Literatur wurden verschiedene Ansätze zur Bestimmung des semantischen Abstands vorgestellt, die sich den Kategorien konstant, manuell und Kantenausgangsgrad zuordnen lassen. Auf der Basis des mit dem semantischen Abstand gewichteten ContextGraph wird der Kontext von Dokumenten mittels einer beschränkten

Kürzesten-Wege-Suche bestimmt. Gegenüber der allgemeinen Berechnung der kürzesten Wege in Graphen konnten im vorliegenden Szenario zwei Einschränkungen festgestellt werden. Die Einschränkung auf Dokumentenknoten kann dabei leicht von jedem n-fach ausgeführten SSSP-Algorithmus – wie z.B. dem n-fachen Dijkstra – ausgenutzt werden. Für die zweite Einschränkung hingegen konnte bewiesen werden, dass kein glatter Algorithmus asymptotischen Nutzen aus der k-Partitheit von Graphen ziehen kann. In die Klasse der glatten Algorithmen fallen dabei alle Algorithmen, deren Laufzeitfunktion aus einer Kombination von z.B. Polynomial-, Logarithmus-, und Wurzelfunktionen zusammengesetzt ist.

Zur Bewertung der Suchresultate wurden im zweiten Teil des Kapitels kontextbasierte Bewertungsmaße für die Relevanz von Dokumenten bezüglich Schlüsselwortanfragen, für die anfrageunabhängige Wichtigkeit von Dokumenten sowie für die Ähnlichkeit von Dokumenten erarbeitet.

Für die Relevanz von Dokumenten bezüglich Schlüsselwortanfragen wurde zunächst das textbasierte $\mathrm{tf.idf}$ Maß vorgestellt und daraus das ausschließlich auf Kontextinformation basierende Maß $\mathrm{ctf.icf}$ hergeleitet. Für die Anwendung des Maßes in Vektorraummodell-basierten Suchmaschinen wurde schließlich eine Kombination beider Maße vorgestellt. Es wurden alternative Ansätze zum Umgang mit booleschen Operatoren im kontextbasierten Szenario vorgestellt, wobei ein besonderes Augenmerk dem Negationsoperator galt. Zur Anwendung des $\mathrm{ctf.icf}$-Maßes auf nicht Vektorraummodell-basierte Suchmaschinen wurden schließlich als Ausblick die jeweiligen Anknüpfungspunkte im probabilistischen Modell genannt.

Zur kontextbasierten Bestimmung der Wichtigkeit von Dokumenten wurde vorgeschlagen, die Berechnung des ObjectRank – einer Erweiterung von Googles PageRank – auf den ContextGraph anzuwenden. Für die Bestimmung der Ähnlichkeit von Dokumenten wurden schließlich die Ansätze der Kozitation und der bibliographischen Kopplung erweitert und auf das kontextbasierte Szenario übertragen. Dazu wurde das zu $\mathrm{ctf.icf}$ alternative Maß $\mathrm{nr.inf}$ eingeführt, bei dem die Ähnlichkeit zweier Dokumente aus der Überlappung ihrer Kontexte berechnet wird.

4

Architektur und Prototyp

Basierend auf den in Kapitel 3 entwickelten Konzepten und entlang der in Abschnitt 1.2 formulierten Anforderungen wurde als Machbarkeitsnachweis eine Architektur entworfen und anhand derer ein prototypisches System implementiert. In diesem Kapitel wird zunächst ein Überblick über die Architektur gegeben. Anschließend wird auf die einzelnen Komponenten eingegangen und einerseits deren Rolle in der Architektur und andererseits deren Implementierung diskutiert, siehe dazu auch [Mangold et al., 2006b; Mangold und Schwarz, 2006].

4.1 Architektur und Überblick

In diesem Abschnitt wird ein Architekturentwurf für eine kontextbasierte Suchmaschine vorgestellt.

Die hier skizzierte Suchmaschine basiert auf dem in Abschnitt 3.1 vorgestellten ContextGraph und erlaubt die Bewertung und Sortierung der Suchergebnisse, entsprechend der in 3.2 diskutierten kombinierten Bewertungsmaße. Um die Anforderung (3.) zur Anpassbarkeit und Erweiterbarkeit zu erfüllen, wurde eine modulare, erweiterbare Systemarchitektur gewählt. Um der Anforderung (5.) in Abschnitt 1.2 bzgl. der Skalierbarkeit des Systems gerecht zu werden, wurde gemäß Abschnitt 2.1.1 einer indexbasierten Suchmaschine gegenüber einer Online-Suchmaschine der Vorzug gegeben. Die Suchmaschine zerfällt demnach logisch in zwei Teile. Der Indexer ist dafür zuständig, relevante Information im Index abzulegen; der Searcher bearbeitet Benutzeranfragen auf Basis der im Index abgelegten Information.

Abbildung 4.1 gibt einen Überblick über den Indexer. In der Darstellung wird zwischen den folgenden Arten von Komponenten unterschieden:

- Kernkomponenten. Die Kernkomponenten realisieren die Grundfunktionalität des Systems. Kernkomponenten sind invariant gegenüber domänenspezifi-

schen Anpassungen und Erweiterungen und sind nur minimal konfigurierbar (z.B. benötigt die Komponente zur Indexerzeugung ein Verzeichnis im Dateisystem zur Ablage des Index).
- Anpassbare Komponenten. Komponenten, die a priori für domänenspezifische Anpassungen ausgelegt sind, werden anpassbar genannt.
- Beispielkomponenten. Domänenspezifische Komponenten, die speziell für bestimmte Szenarien ausgelegt sind, werden als Beispielkomponenten bezeichnet.

Im Folgenden wird ein Überblick über die Funktionalitäten der einzelnen Komponenten des Indexer gegeben.

Die *Import-Komponenten* übersetzen jeweils Information aus strukturierten oder semistrukturierten Datenquellen in einen ContextGraph. Beim Import von relationalen Datenbanken kann dabei vorgegangen werden, wie in Abschnitt 3.1.2 beschrieben. Zum Import von Daten aus Ontologien oder aus semistrukturierten Datenquellen muss jeweils eine eigene Importkomponente zur Verfügung gestellt werden.

Die *ContextGraph-Komponente* enthält eine Implementierung der in Definition 3.1.1 festgelegten Struktur.

Die Komponente zur *Kantengewichtung* weist den Kanten des ContextGraphs Gewichte für den semantischen Abstand zu. Wie in Abschnitt 3.1.3.2 beschrieben, können bei der Kantengewichtung verschiedene Strategien verfolgt werden, die von der Zuweisung konstanter Werte für alle Kanten des Graphen über die manuelle, evtl. schemabasierte Zuweisung bis zur vollautomatischen Zuweisung reichen. Die Implementierung der Kantengewichtungskomponente kann somit entweder allgemeingültig oder domänenspezifisch abhängig vom jeweiligen ContextGraph sein.

Die Komponente zur Berechnung der *ObjectRank*-Werte berechnet eine kontextbasierte, anfrageunabhängige Wichtigkeit der im ContextGraph erfassten Dokumente. Die Berechnung der ObjectRank-Werte wurde in Abschnitt 3.2.2.3 beschrieben. Im Rahmen der Architektur ist die Object-Rank-Komponente als Beispielkomponente zur Berechnung eines kontextbasierten Wichtigkeitsmaßes zu sehen. Sie könnte leicht ersetzt oder erweitert werden durch eine Komponente, die z.B. die Wichtigkeit von Dokumenten basierend auf dem Datum der letzten Änderung oder dem zugeordneten Unternehmensbereich berechnet.

Die Berechnung der *kürzesten Wege* liefert für jeden Dokumentenknoten des ContextGraphs alle Knoten seines Kontexts sowie den jeweiligen semantischen Abstand entlang des kürzesten Weges zwischen Dokumentenknoten und Kontextknoten. Das Ergebnis der Kürzesten-Wege-Komponente bildet somit die Grundlage zur Berechnung kontextbasierter Relevanz, d.h. der Relevanz von Begriffen für Dokumente in deren Kontext sie enthalten sind.

Der Text der im ContextGraph verzeichneten Dokumente wird von der *Dokumentenimport*-Komponente geladen. Zu einer gegebenen URL wird das zugehörige Dokument geladen und ins Text-Format konvertiert. Dieser Text bildet die Grundlage zur Berechnung der Relevanz der im Text enthaltenen Begriffe für das jeweilige

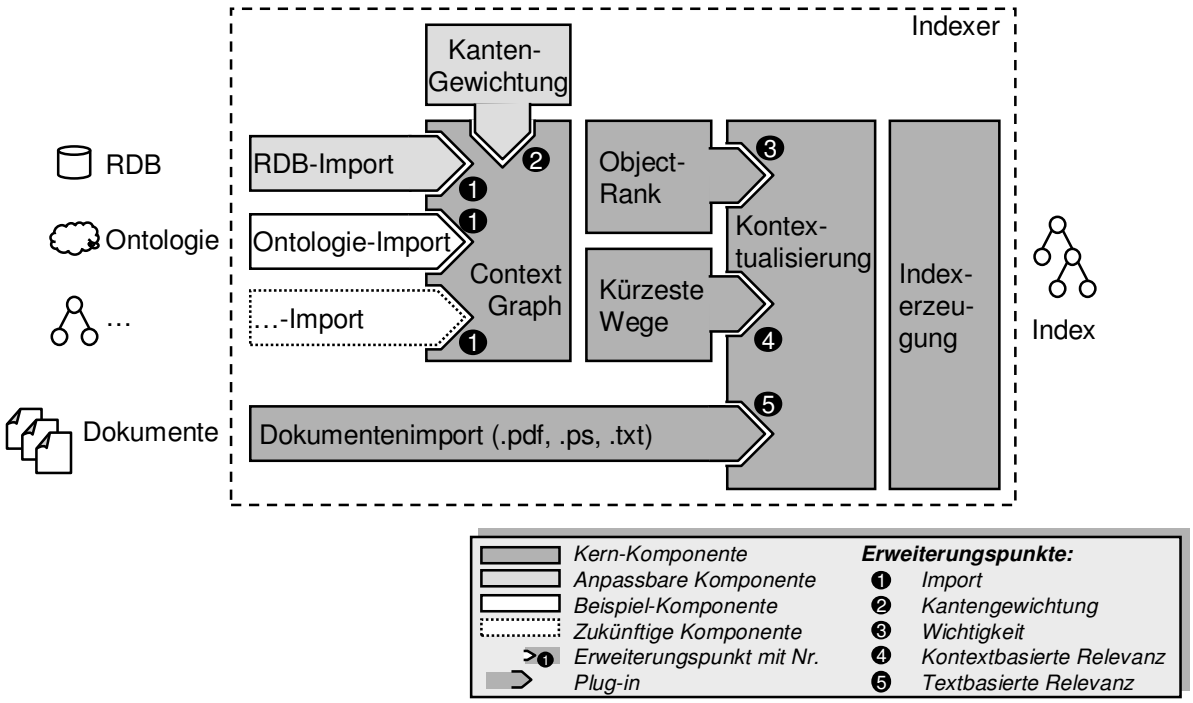


Abbildung 4.1: Indexer.

Dokument.

Die *Kontextualisierungskomponente* integriert die Informationen zur Wichtigkeit und zur kontextbasierten Relevanz mit denen der textbasierten Relevanz und propagiert sie and die Komponente zur Indexerzeugung.

Die Komponente zur *Indexerzeugung* ist dafür verantwortlich, die von der Kontextualisierungskomponente erhaltene Information in einem Index abzulegen, welcher zur Anfragezeit eine effiziente Dokumentenrecherche ermöglicht. Der Aufbau des Index muss die Bewertung von Suchresultaten unterstützen.

Im Gegensatz zum allgemeinen Aufbau indexbasierter Suchmaschinen nach Abbildung 2.1 enthält der Protoyp keine dedizierte Crawler-Komponente. Anstelle dessen wird die für die Suchmaschine relevante Information per Konfigurationsdatei spezifiziert und durch die Import-Komponenten geladen. Für einen weiteren Ausbau der Suchmaschine ist die Erweiterung der Architektur um eine Crawler-Komponente sinnvoll, da so dann auch automatisiert inkrementelle Aktualisierungen des Index vorgenommen werden können.

Abbildung 4.2 zeigt die Architektur des Searchers.

Der Benutzer interagiert mit dem System per Web Browser. Der Browser greift über den in der Abbildung nicht dargestellten Web Server auf die Komponente der *Benutzerschnittstelle* zu. Diese leitet die Benutzeranfrage an die Komponente zur semantischen Suche weiter. Wird von dort ein Suchresultat zurückgegeben, so wird dies zur Darstellung im Web Browser aufbereitet und dem Benutzer angezeigt.

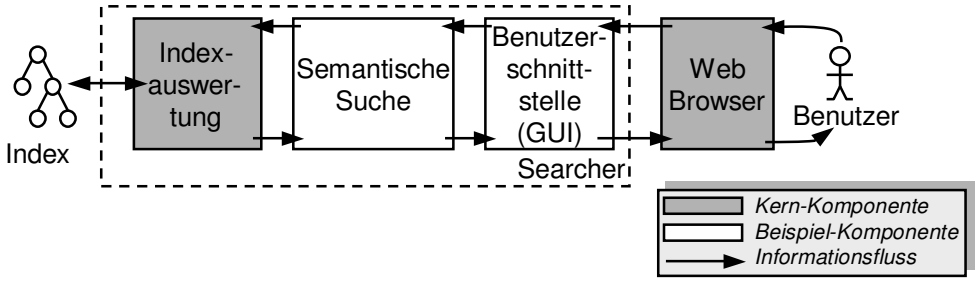


Abbildung 4.2: Searcher.

Die Komponente zur *semantischen Suche* ist für die Realisierung von semantischer Suchfunktionalität, wie sie in Abschnitt 2.5 diskutiert wurden, vorgesehen. Sie erhält die Anfrage des Benutzers und kann diese bspw. ontologiebasiert modifizieren oder in mehrere Anfragen aufteilen. Das Ergebnis dieser möglicherweise mehreren Anfragen kann dann von derselben Komponente ebenfalls modifiziert, evtl. gefiltert oder entsprechend der Modifikation der Anfrage bearbeitet werden, bevor es an die Benutzerschnittstellenkomponente weitergeleitet wird.

Nachdem die Benutzeranfrage von der Komponente zur semantischen Suche an die Komponente zur *Indexauswertung* weitergeleitet wurde, verwendet diese den Index zur Bearbeitung der Anfrage. Da von der Indexerzeugungskomponente Informationen sowohl aus dem Text als auch aus dem Kontext der erfassten Dokumente im Index abgelegt wurden, werden bei der Indexauswertung ebenfalls beide Arten von Informationen erfasst. Die Komponente zur Indexauswertung ist ebenfalls für die Bewertung der Suchresultate zuständig.

Diskussion Die Architektur realisiert eine *indexbasierte* Suchmaschine. Gegenüber der indexbasierten Suchmaschine hat eine Online-Suchmaschine den Vorteil, dass die recherchierte Information stets konsistent ist mit dem aktuellen Stand der Information. Demgegenüber steht der Nachteil, dass sowohl die Performanz der Suchmaschine als auch die Performanz der eingesetzten Datenbank deutlich in Mitleidenschaft gezogen werden können.

Für eine kontextbasierte Online-Suchmaschine findet die Suche sowohl im Text als auch im Kontext der Dokumente zur Anfragezeit statt. Als Realisierung käme etwa die Schlüsselwortsuche auf einer Datenbank in Frage, in der die zu durchsuchenden Dokumententexte als Attributwerte abgelegt sind. Zur Bestimmung des Dokumentenkontexts muss dann die Berechnung der kürzesten Wege bzw. ein alternativer Spreading-Activation-Algorithmus ebenfalls zur Laufzeit durchgeführt werden, was insgesamt keine zufriedenstellende Performanz erwarten lässt.

Das in diesem Kapitel vorgestellte System ist als Prototyp und als Versuchsplattform für die kontextbasierte Suche zu sehen. Die Funktionalitäten wurden dabei auf wesentliche Aspekte zur Erweiterung der textbasierten Suche auf das kontextbasierte Szenario beschränkt. Viele im Bereich der textbasierten Suche gängige Funktionali-

täten wurden dabei nicht in Betracht gezogen und nicht implementiert. Hierfür seien die folgenden beiden Beispiele genannt:

- Explizite Metadaten. Viele Suchmaschinen bieten dem Benutzer die Möglichkeit, nur auf ausgewählten Metadaten von Dokumenten, wie z.B. Titel, Autoren, etc. zu suchen. Diese Funktionalität kann z.B. in Lucene leicht mit dem Konzept der L-Fields realisiert werden. Analog dazu könnte auch die kontextbasierte Suche mit geringem Aufwand um diese Funktionalität erweitert werden, siehe etwa [Sinha und Karger, 2005].

- Dokumentstruktur. Textbasierte Suchmaschinen verwenden oftmals die innere Struktur von Dokumenten, um die Relevanz von Begriffen besser bestimmen zu können. So werden Begriffe zu Beginn des Dokuments, in (Unter-) Überschriften oder in kursiver Schrift höher bewertet als andere. Diese Konzepte könnten bei der kombinierten Berechnung der Begriffsgewichte direkt mitberücksichtigt werden.

Bei einer Weiterentwicklung des Prototypen können alle derartigen Erweiterungen problemlos in das System integriert werden.

In den nächsten Abschnitten werden die einzelnen Komponenten der Architektur sowohl hinsichtlich ihrer Funktionalität innerhalb der Architektur als auch hinsichtlich ihrer Implementierung vorgestellt. Für die Realisierung des Prototyps wurde die Programmiersprache Java[1] eingesetzt, die seit 1995 von der Firma Sun Microsystems entwickelt wird und sich in den letzten Jahren zu einer weitverbreiteten Programmiersprache im Bereich der Web- und Datenbankprogrammierung etabliert hat. Die Programmiersprache wurde dabei nicht nur wegen ihrer Plattformunabhängigkeit, sondern auch wegen der großen Anzahl an frei verfügbaren Programmbibliotheken ausgewählt.

4.2 Import und ContextGraph

In diesem Abschnitt werden die Komponenten zum Import strukturierter Daten und die ContextGraph-Komponente eingeführt. Zunächst wird jedoch auf zwei für die Implementierung der Import-Komponenten wichtige Technologien eingegangen.

4.2.1 Eingesetzte Bibliotheken

Das Semantic-Web-Framework Jena[2] der Firma Hewlett-Packard ist eine weitverbreitete Java-Bibliothek zur Manipulation von OWL- und RDF-Ontologien. Da manche Ontologie-Editoren, wie z.B. Protégé[3], Jena als Back-End zum Abspeichern von

[1] `http://java.sun.com/` [29.04.2006]
[2] `http://jena.sourceforge.net/` [29.05.2006]
[3] `http://protege.stanford.edu/` [12.07.2006]

Ontologien verwenden, ist jeweils sichergestellt, dass Ontologien, die einem dieser Editoren entstammen, von Jena eingelesen und verarbeitet werden können.

Eine weitverbreitete Anwendung des Jena-Frameworks besteht darin, Ontologien zu verarbeiten, welche in der XML-Syntax von OWL vorliegen. Im hier vorgestellten Prototyp wird das Jena-Framework dazu eingesetzt, Ontologien auf einfache Weise in den ContextGraph zu überführen und auf diese Weise deren Inhalte der Suchmaschine zur Verfügung zu stellen.

Die D2RQ[4]-Bibliothek leistet die Abbildung relationaler Datenbanken auf Ontologien und arbeitet dabei als Back-End von Jena. Eine mittels D2RQ an Jena angeschlossene Datenbank kann somit über Jena als OWL- oder RDF-Ontologie angesprochen werden. Dabei werden in der Regel Tabellen der Datenbank auf Konzepte in der Ontologie abgebildet; Tupel werden zu Instanzen, vgl. [Bizer und Seaborne, 2004].

In der im Protoyp eingesetzten Version[5] müssen die Konzepte und Relationen, auf die die Datenbank abgebildet werden soll, im System explizit konfiguriert werden. Eine Weiterentwicklung von D2RQ macht diese Vorgabe überflüssig [Cyganiak und Bizer, 2006]. Damit können relationale Datenbanken vollautomatisch als OWL-Ontologie dargestellt werden.

Der Import relationaler Datenbanken in den ContextGraph via D2RQ und der OWL-Schnittstelle von Jena konnte somit mit minimalem Implementierungsaufwand durchgeführt werden. Die Darstellung von Datenbanken als Ontologie ermöglicht zusätzlich die Integration verschiedener Informationsquellen im ContextGraph mit Mechanismen der Ontologieintegration, siehe dazu etwa [Euzenat, 2004; Pinto und Martins, 2001; Noy und Musen, 2000].

4.2.2 Import-Komponenten

Abbildung 4.1 zeigt drei Arten möglicher Import-Komponenten, die sich in der Art der importierten Datenstruktur unterscheiden. Der RDB-Import wird gemäß der in Abschnitt 3.1.2 beschriebenen Methode durchgeführt. Zu diesem Zweck wurden zwei funktional identische, jedoch auf technologischer Ebene unterschiedliche Komponenten implementiert.

Die *ontologiebasierte* Variante der RDB-Import-Komponente verwendet die D2RQ-Bibliothek zur Darstellung von Datenbanken als Ontologie. D2RQ wird dazu als Backend der Jena-Bibliothek eingesetzt, was den Zugriff auf die so entstandene Ontologie über die Jena-Schnittstelle ermöglicht. Der Vorteil dieser ontologiebasierten Variante liegt in ihrer leichten Anpassbarkeit. Zur Übertragung auf andere Datenbanken muss lediglich die Konfiguration der D2RQ-Bibliothek entsprechend modifiziert

[4] `http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/` [12.07.2006]

[5] D2RQ Version 0.2

werden. Der Nachteil der ontologiebasierten Variante liegt in ihrer mangelnden Skalierbarkeit. Da die ontologiebasierte Variante ab einer Graphgröße von ca. 50 000 Kanten deutliche Performanzprobleme aufweist, wurde die funktional identische *direkte* RDB-Import-Komponente implementiert. Die direkte Variante ist zugeschnitten auf ein spezielles Datenbankschema und kann daher nur Datenbanken importieren, die diesem entsprechen.

Die Komponente zum Ontologie-Import entspricht dem Frontend der ontologiebasierten Datenbank-Import-Komponente. Über die Jena-Bibliothek kann hiermit auf beliebige Ontologien im RDF- oder OWL-Format zugegriffen werden. Konzepte und Instanzen der Ontologie werden im ContextGraph auf Knoten abgebildet, Relationen werden durch Kanten repräsentiert.

Analog zu den Komponenten für den Import von relationalen Datenbanken und Ontologien sind beliebige weitere Komponenten zum Import semistrukturierter oder strukturierter Daten denkbar.

4.2.3 Die ContextGraph-Komponente

Die ContextGraph-Komponente enthält eine Implementierung der in Definition 3.1.1 festgelegten Struktur. Um den Speicheraufwand zu reduzieren, werden Kanten mit gleicher Beschriftung und gleichem Abstand zu Kantentypen zusammengefasst und Beschriftung sowie Abstand dem Typen zugeordnet.

Um verschiedene Algorithmen zur Bestimmung der kürzesten Wege im ContextGraph testen zu können, muss je nach Algorithmus die Knotendatenstruktur erweitert werden. Zur Realisierung des Dijkstra-Algorithmus benötigen die Knoten z.B. ein Attribut zur Speicherung des Abstands zum aktuellen Ursprungsknoten. Der HiddenPath-Algorithmus hingegen setzt pro Knoten eine Liste der optimalen Eingangskanten voraus.

4.3 Kantengewichtung

Die Komponente zur Bestimmung der Kantengewichte kann, wie in Abschnitt 3.1.3.2 beschrieben, auf verschiedene Arten realisiert werden. Im vorliegenden Protoyp wird der manuelle Ansatz verfolgt, der davon profitiert, dass der ContextGraph dem Schema der Datenbank gehorcht. Es genügt daher, eine einfache, schemabasierte Abbildung der Kantenbeschriftung auf den semantischen Abstand vorzunehmen. Kanten mit gleicher Beschriftung erhalten dann stets dasselbe Gewicht. Die Zuordnung von Gewichten zu Kanten lässt sich per Konfigurationsdatei festlegen.

Die Kantengewichtungskomponente im Prototyp setzt das folgende einfache Gewichtungsschema um: Alle Kanten, die aus Fremdschlüsselbeziehungen stammen, erhalten Gewicht 1 und alle Kanten, die eine Tupelzugehörigkeit anzeigen, Gewicht 0. Kanten, die von Tupelknoten zu Tabellenknoten zeigen, erhalten ebenfalls das Ge-

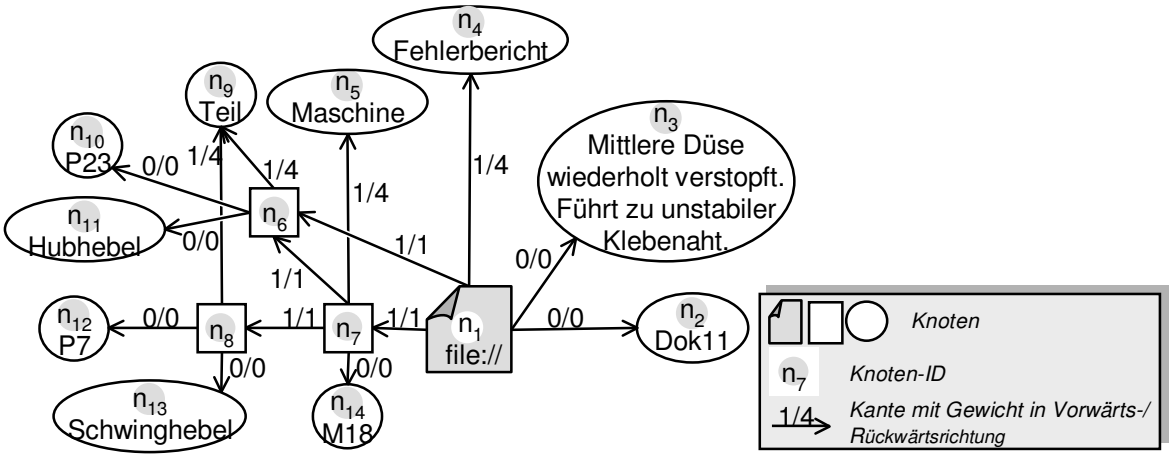


Abbildung 4.3: Gewichtung der Kanten.

wicht 1, die ihnen entgegengesetzt ausgerichteten Kanten von Tabellenknoten zu Tupelknoten erhalten Gewicht 4, siehe dazu die Diskussion in Abschnitt 3.1.3. Die Gewichtung der Kanten des Beispiels ist in Abbildung 4.3 dargestellt. Zur Erleichterung der weiteren Diskussion wurde die ursprüngliche Abbildung 3.1 hier um Knoten-IDs erweitert.

4.4 ObjectRank, Kürzeste Wege und Dokumentenimport

Die Berechnung des ObjectRank-Werts für die Knoten des ContextGraphs wird durchgeführt, wie in Abschnitt 3.2.2.3 beschrieben. Die von der Object-Rank-Komponente an den Kontextualisierer übergebene Datenstruktur ist demnach denkbar einfach. Für das Beispiel ist sie in Abbildung 4.4 (a) dargestellt. Die Knoten-ID steht hier, wie auch im Folgenden, stellvertretend für das Objekt, welches mit dieser Knoten-ID identifiziert wird.

Zur Berechnung der kürzesten Wege wurden verschiedene Algorithmen implementiert. Der APSP-Algorithmus nach Floyd-Warshall [Cormen et al., 1994, S.558] ist einfach zu implementieren, hat jedoch kubische Laufzeit. Es wurde zusätzlich ein beschränkter n-facher Dijkstra implementiert, der, wie in Abschnitt 2.7.2.1 beschrieben, nur die kürzesten Wege innerhalb einer beschränkten Umgebung des Startknoten berechnet. Um diesen Algorithmus weiter zu beschleunigen, wurden nicht alle Knoten des ContextGraphs als Startknoten verwendet, sondern lediglich die Dokumentenknoten. Schließlich wurde noch ein beschränkter HiddenPath-Algorithmus nach Abschnitt 2.7.2.2 implementiert. Die Analyse der Laufzeiten des beschränkten n-fachen Dijkstra auf Dokumentenknoten und des beschränkten HiddenPath finden sich in Abschnitt 5.4.

Bei der Implementierung des Algorithmus nach Dijkstra kommt als Implementierung der Prioritätswarteschlange eine hauptspeicherbasierte Heap-Implementierung

zum Einsatz. Beim HiddenPath-Algorithmus muss die Prioritätswarteschlage deutlich mehr Datensätze aufnehmen, da beim HiddenPath die Dijkstra-Algorithmen logisch parallel ablaufen. Daher wurde hierfür eine externspeicherbasierte Prioritätswarteschlange implementiert, die auf Javas NIO-Technologie basiert. Diese Implementierung nutzt aus, dass die Pfade im Graphen stets ganzzahliges Gewicht haben, und nur eine beschränkte Anzahl unterschiedlicher Gewichte betrachtet werden. Für jedes Gewicht wird daher eine eigene Datenstruktur angelegt, welche teilweise im Hauptspeicher und teilweise im Dateisystem abgelegt ist. Darauf aufbauend wurde ein einfacher Caching-Mechanismus entwickelt, der die Anzahl der Zugriffe auf den Sekundärspeicher minimiert.

Als Datenstruktur zur Speicherung der Resultate kommt ein Array von Hash-Tabellen (java.util.HashMap) zum Einsatz, wobei die Länge des Array von der Anzahl der Knoten im ContextGraph abhängt. Jedem Knoten x wird auf diese Weise eine Hash-Tabelle zugeordnet, in der zu jedem Knoten y im Kontext von x ein Pfadobjekt hinterlegt ist. Ein Pfadobjekt besteht dabei aus dem Startknoten x, dem zweiten Knoten des Pfads, dem Endknoten y sowie dem Gewicht des Pfads. Auf diese Weise lässt sich einerseits für gegebene x und y effizient auf den Pfad zwischen x und y zugreifen. Andererseits ist es ebenfalls möglich, effizient den gesamten Kontext eines Knoten x zu erfragen.

Die Ausgabe der Komponente zur Berechnung der kürzesten Wege besteht aus einer Liste der Knoten aller zu indexierenden Dokumente. Jedem solchen Dokumentenknoten ist eine Liste aller Knoten aus dessen Kontext sowie der jeweilige semantische Abstand zugeordnet. Für das Dokument unseres Beispiels ist die zugehörige Datenstruktur in Abbildung 4.4 (b) dargestellt. Für die spätere Diskussion enthält die Abbildung zusätzlich die Werte für die semantische Nähe sim_3 sowie, der Übersichtlichkeit halber, ebenfalls die Knotenbeschriftung. Letztere ist nicht direkt in der Datenstruktur enthalten, sondern nur indirekt über die Knotenreferenz erreichbar.

Die Komponente zum Dokumentenimport erhält von der Kontextualisierungskomponente eine URL und ist dafür verantwortlich, den Inhalt des entsprechenden Dokuments zu beschaffen. Ihre Ausgabe ist beispielhaft in Abbildung 4.4 (c) dargestellt. Für den Protoyp wurde eine einfache Import-Komponente für unformatierte Text-Dateien erstellt. Der Import von Dokumenten im PDF-Format basiert auf der Java-Bibliothek PDFBox[6], die das Lesen, Schreiben und Bearbeiten von PDF-Dokumenten unterstützt. Da die Konvertierung von Dateien durch die PDFBox-Bibliothek sehr zeitintensiv ist, wurden für die Evaluierung des Prototyps in einem Batch-Programm alle Dateien vor der Indexierung konvertiert und im Textformat abgelegt, sodass die Messungen der Indexierungszeit nicht von der Laufzeit der Konvertierungsbibliothek beeinflusst werden.

[6] `http://www.pdfbox.org/` [08.08.2006]

❸

Knoten-ID	Wichtigkeit
n_1	0.8

(a)

❹

Knoten-ID	Knoten-ID	dist	sim_3	Knoten-Beschriftung
n1	n_2	0	3	Dok11
	n_3	0	3	Mittlere Düse wiederholt verstopft. Führt zu unstabiler Klebenaht
	n_4	1	2	Fehlerbericht
	n_5	2	1	Maschine
	n_9	2	1	Teil
	n_{10}	1	2	P23
	n_{11}	1	2	Hubhebel
	n_{12}	2	1	P7
	n_{13}	2	1	Schwinghebel
	n_{14}	1	2	M18

(b)

❺

URL	Text
file:///Z:/docs/fb/Doc11.txt	Es taucht wiederholt das Problem auf, dass die mittlere Düse verstopft. Dies führt v.a. links zu unscharfer Klebenaht. Beim Aufbringen zeigt der Kleber …

(c)

Abbildung 4.4: Beispieldaten an den Schnittstellen der Kontextualisierungskomponente. (a) ❸, (b) ❹ und (c) ❺.

4.5 Kontextualisierung

Die Kontextualisierungskomponente bildet den Kern des Indexers. Ihre Aufgabe besteht darin, für jedes Dokument die vorhandene Information zu sammeln, zu integrieren und an die Komponente zur Indexerzeugung weiterzugeben.

Zum Sammeln der dokumentbezogenen Information verfügt die Kontextualisierungskomponente über drei Schnittstellen. Über die Schnittstelle ❸ für die kontextbasierte Wichtigkeit von Dokumenten wird im Prototyp das Ergebnis der ObjectRank-Komponente eingelesen. Grundsätzlich können hier jedoch die Ergebnisse weiterer, ggf. domänenspezifischer Komponenten aufgenommen werden. Über die Schnittstelle zur kontextbasierten Relevanz ❹ wird für ein Dokument die Relevanz – d.h. in diesem Falle die semantische Nähe – aller im Kontext des Dokuments enthaltenen Begriffe übergeben. Analog dient die Schnittstelle zur textbasierten Relevanz ❺ dazu, alle im Text eines Dokuments enthaltenen Begriffe an die Kontextualisierungskomponente zu übermitteln. Wie Abbildung 4.4 (c) zeigt, beinhaltet dies keine Bewertung sondern lediglich die Aufzählung aller Begriffe des Dokumententexts.

4.6 Indexerzeugung und Indexauswertung

Die Komponente zur Indexerzeugung ist dafür verantwortlich, die von der Kontextualisierungskomponente erhaltene Information in einem Index abzulegen, der zur Anfragezeit von der Indexauswertungskomponente gelesen werden kann.

Der Index muss dabei so strukturiert werden, dass Kontextinformation im Index abgelegt und bei der Bewertung der Suchresultate entsprechend berücksichtigt werden kann. Diese Strukturierung des Index wird bei der Indexerzeugung angelegt und

zur Anfragezeit in der Komponente zur Indexauswertung berücksichtigt.

Im Rahmen dieser Arbeit wurden drei alternative Implementierungen jeweils für Indexerzeugung und Indexauswertung entworfen und umgesetzt, die im folgenden Einzelschrittindex, Doppelschrittindex für Schlüsselwortsuche und Doppelschrittindex für Ähnlichkeitssuche genannt werden. Die jeweils diskutierte Umsetzung der Rankingstrategie beschreibt den im Protoyp gewählten Ansatz. Es versteht sich von selbst, dass jeweils alternative Ansätze denkbar wären.

Alle drei Implementierungen basieren auf der Lucene, einer Java-Bibliothek zur Volltextsuche. Die im Protoyp gewählte Umsetzung hatte zum Ziel, nicht in die Lucene-interne Anfrageverarbeitung und Resultatsbewertung einzugreifen und so die guten Performanz-Eigenschaften der reinen Textsuche weitestgehend zu erhalten.

In diesem Abschnitt werden zunächst die für die vorliegende Arbeit wichtigen Eigenschaften von Lucene erläutert. Anschließend werden der Einzelschrittindex, der Doppelschrittindex für Schlüsselwortsuche und der Doppelschrittindex für Ähnlichkeitssuche als Alternativen zur Strukturierung des Index vorgestellt.

4.6.1 Lucene

Das Lucene[7]-Framework der Apache Software Foundation ist eine Java-Bibliothek zur Volltextsuche auf Dokumenten [Gospodnetić und Hatcher, 2005]. Lucene beinhaltet Klassen zur Dokumentenanalyse (z.B. Entfernung von Stoppworten und Wortstammbestimmung), zur Ablage von Dokumenten in einem Index und zur Verarbeitung von Schlüsselwortanfragen basierend auf einem solchen Index. Lucene ist dabei keine fertige Lösung zur Volltextsuche auf beliebigen Dokumentenstämmen, sondern beinhaltet lediglich die dafür notwendigen Datenstrukturen und Algorithmen. Die Erstellung einer primitiven Textsuchmaschine, basierend auf Lucene, erfordert jedoch nur wenige Zeilen Code.

Für die vorliegende Arbeit sind vor allem die systeminterne Modellierung indexierbarer Dokumente, die Freiheitsgrade beim Erstellen von Indexen und die Bewertung von Suchresultaten von Interesse. Auf jeden dieser Bereiche wird im Folgenden kurz eingegangen.

4.6.1.1 Dokumente

Dokumente der realen Welt werden in Lucene durch die Datenstruktur “Document” modelliert. Um Verwirrung zu vermeiden, wird diese Lucene-eigene Datenstruktur im Folgenden mit L-Document bezeichnet.

Logisch repräsentiert ein L-Document eine Einheit der realen Welt, welche im Index abgelegt und dort mittels Schlüsselwortsuche wiedergefunden werden kann. Intern besteht ein L-Document aus einer Liste von Feldern, welche im Folgenden

[7] `http://lucene.apache.org/` [12.07.2006]

L-Fields genannt werden. Jedes L-Field besteht aus einem Namen und einem dazugehörigen Text. In einer Standardsuchmaschine entspricht bspw. ein L-Document einem realen Dokument. Es könnte aus L-Fields mit Namen Ueberschrift, Text, Autor und URL zusammengesetzt sein, welche die Überschrift, den Text, den Autor und die URL des Dokuments enthalten. Eine solche Aufteilung ermöglicht dann z.B. die Suche nach Dokumenten, die die Suchbegriffe in ihrem L-Field mit Namen Ueberschrift enthalten. Die Lucene-eigene Notation einer solchen Suche nach dem Begriff Klebenaht in den Überschriften aller Dokumente lautet dann: Ueberschrift:Klebenaht.

4.6.1.2 Erstellung des Index

Als Indexstruktur setzt Lucene invertierte Listen ein, welche in proprietärem Dateiformat auf dem Sekundärspeicher abgelegt werden. Eine detaillierte Beschreibung der Indexstrukturen ist bei Gospodnetić und Hatcher [2005] zu finden. Beim Indexieren können für jedes L-Field die folgenden drei Parameter festgelegt werden:

- Stored. Der Stored-Parameter legt fest, ob der Textinhalt eines L-Fields aus dem Index gelesen können werden soll. Typischerweise müssen z.B. URLs von Dokumenten aus dem Index gelesen werden können, um das ursprüngliche Dokument nach dem Rechercheprozess zu lokalisieren.
- Indexed. Der Indexed-Parameter legt fest, ob nach dem Textinhalt eines L-Fields gesucht werden können soll. Die URL von Dokumenten könnte bspw. nicht indexiert werden, während nach ihrer Überschrift oder ihrem Autor in der Regel gesucht werden können soll.
- Analyzed. Der Analyzed-Parameter legt fest, ob der Textinhalt eines L-Fields der Textanalyse, d.h. Wortstammbestimmung, Entfernung von Stoppworten etc. unterzogen werden soll. Nicht analysiert werden in der Regel L-Fields, die URLs oder spezielle Buchstaben-Zahlen-Kombinationen, wie z.B. Telefonnummern, enthalten.

4.6.1.3 Bewertung von Suchresultaten

Zur Bewertung der Suchresultate verwendet Lucene eine leicht modifizierte Vektorraum / tf.idf-Strategie. Um dem Anwender eine erweiterte Kontrolle über die Bewertung von Suchergebnissen zu geben, bietet Lucene die Möglichkeit, Treffer in L-Fields oder in L-Documents speziell zu belohnen. Diese Möglichkeit wird Boost genannt und kann folgendermaßen angewendet werden:

- Document-Boost. Werden in einem Szenario manche Dokumente als global wichtiger eingeschätzt als andere, vgl. Abschnitt 2.1.3.3, so kann zur Indexierungszeit festgelegt werden, dass solche Dokumenten besser bewertet werden.
- Field-Boost. Analog zum Document-Boost kann ebenfalls zur Indexierungszeit ein Field-Boost festgelegt werden, der Treffer auf einzelnen L-Fields belohnt. So lässt sich bspw. festlegen, dass Treffer auf Überschriften schwerer

wiegen als auf normalem Text.

- Term-Boost. Wie beim Field-Boost zur Indexierungszeit lassen sich zur Anfragezeit Treffer einzelner Begriffe belohnen. Ist es für einen Benutzer besonders wichtig, dass der Begriff Klebenaht im gefundenen Dokument vorkommt und soll nach Möglichkeit auch Schwinghebel vorkommen, so werden in der Ergebnismenge der Anfrage Klebenaht^2 Schwinghebel Dokumente, die den Begriff Klebenaht enthalten, durch den Term-Boost ^2 doppelt so relevant eingestuft, wie Dokumente, die den Begriff Schwinghebel enthalten.

Die folgenden Formeln zur Umsetzung des Vektorraummodells und der Begriffsgewichte in Lucene basieren auf einer Analyse des Source-Codes[8] von Lucene, da weder die Java-Dokumentation[9] noch das Lucene-Buch [Gospodnetić und Hatcher, 2005] an dieser Stelle vollständige Angaben liefern.

In den Formeln wird ein L-Document d und eine Anfrage q jeweils als eine Menge von L-Fields und ein L-Field f als eine Menge von Begriffen t betrachtet. Um die Darstellung zu erleichtern werden teilweise L-Documents und Anfragen direkt als die Menge der Begriffe aller darin enthaltenen L-Fields betrachtet; die Bedeutung wird jeweils durch die Wahl der Bezeichner klar.

Die Relevanz $\mathrm{rel}(d, q)$ eines L-Documents d bzgl. einer Anfrage q wird ähnlich zum in Abschnitt 3.2.1.2 vorgestellten Skalarprodukt mit folgender Formel aus den Begriffsgewichten (term-weights) qtw der Anfrage und den Begriffsgewichten dtw des jeweiligen L-Documents berechnet.

$$\mathrm{rel}(d, q) = \sum_{t \in q} \left(\mathrm{dNorm}(q, d, t) \cdot \sum_{f \in d} \mathrm{qtw}(q, f, t) \cdot \mathrm{dtw}(d, f, t) \right) \tag{4.1}$$

Dabei ist die Dokumenten-Norm $\mathrm{dNorm}(q, d, t)$ definiert als der Quotient aus der Anzahl der L-Fields f des L-Documents d, die den Suchbegriff t enthalten, und der Anzahl der L-Fields, in denen t gesucht wird: $\mathrm{dNorm}(q, d, t) = \frac{|\{f \in d \mid t \in f\}|}{|\{f \in q \mid t \in f\}|}$. Die Begriffsgewichte der Anfrage $\mathrm{qtw}()$ und die des Dokuments $\mathrm{dtw}()$ sind wie folgt definiert:

$$\mathrm{qtw}(q, f, t) = \mathrm{idf}(f, t) \cdot \mathrm{tb}(f, t) \cdot \mathrm{qNorm}(q)$$

$$\mathrm{dtw}(d, f, t) = \mathrm{tf}(f, t) \cdot \mathrm{idf}(f, t) \cdot \mathrm{fb}(f) \cdot \mathrm{db}(d) \cdot \mathrm{fNorm}(f, d)$$

Dabei ist

[8] Version 1.4.3

[9] `http://lucene.apache.org/java/docs/api/index.html` bzw. `http://lucene.apache.org/java/docs/api/org/apache/lucene/search/Similarity.html` [13.07.2006]

- $\mathrm{idf}(f,t)$ die inverse Dokumentenhäufigkeit, die nach folgender Formel berechnet wird: $\mathrm{idf}(f,t) = \log(\frac{D}{|\{d \mid t \in d.f\}|+1}) + 1$, wobei $d.f$ das L-Field f des L-Documents d bezeichnet und D wieder für die Anzahl aller Dokumente steht,

- $\mathrm{tb}(f,t)$ der oben eingeführte Term-Boost,

- $\mathrm{qNorm}(q)$ eine Normierung der Begriffsgewichte der Anfrage, die ermöglicht, dass die Bewertungen unterschiedlicher Anfragen vergleichbar sind und nach folgender Formel berechnet wird: $\mathrm{qNorm}(q) = \frac{1}{\sqrt{\sum_{f \in q} \sum_{t \in f} (\mathrm{idf}(f,t) \cdot \mathrm{tb}(f,t))^2}}$,

- $\mathrm{tf}(f,t)$ die Begriffshäufigkeit gemäß der Formel $\mathrm{tf}(f,t) = \sqrt{\mathrm{freq}(t,f)}$,

- $\mathrm{fb}(f)$ der Field-Boost von f,

- $\mathrm{db}(d)$ der Document-Boost von d und

- $\mathrm{fNorm}(f,d)$ eine Normierung des Begriffsgewichts des Dokuments entsprechend der Formel $\mathrm{fNorm}(f,d) = \frac{1}{\sqrt{|d.f|}}$, wobei $|d.f|$ die Anzahl der Begriffe im L-Field f von L-Document d bezeichnet.

Im Vergleich mit dem in Abschnitt 3.2.1.2 eingeführten Vektorraummodell ergibt sich somit ein wesentlicher Unterschied der von der Existenz der L-Fields herrührt. Um diese zu behandeln, baut Lucene einen Vektorraum auf, dessen Dimensionalität sich nicht an der Anzahl der zu indexierenden Begriffe orientiert, sondern an dieser Zahl multipliziert mit der Anzahl der zu indexierenden L-Fields. Zur Relevanzbestimmung wird ein L-Document also logisch als ein Vektor angesehen, in dem für jede Kombination aus L-Field und Begriff ein durch $\mathrm{tf.idf}$ berechneter Eintrag existiert.

Neben dieser grundlegenden Erweiterung weist das Lucene-Bewertungsschema noch einige kleinere Unterschiede zur in Abschnitt 3.2.1.2 eingeführten Berechnung auf. So basiert z.B. bei der Berechnung der Begriffsgewichte des Dokuments die Normierung von tf (welche durch die Funktion $\mathrm{fNorm}()$ realisiert wird) nicht auf der maximalen Häufigkeit der Begriffe im Text, sondern vielmehr auf der Länge des Texts. Weitere Unterschiede sind z.B in der Berechnung des Ähnlichkeitsmaßes zwischen Vektoren und in der modifizierten Definition von idf zu finden. Diese Unterschiede sind ein Beispiel dafür, dass in der Praxis viele Variationen des in Abschnitt 3.2.1.2 vorgestellten Vektorraummodells existieren. Dabei ist zu beachten, dass beim Entwurf eines Bewertungsmaßes neben den logischen bzw. semantischen Beweggründen stets auch die Auswirkungen auf die Performanz eines Systems beachtet werden müssen.

4.6.2 Einzelschrittindex

Der Einzelschrittindex bildet den Anfragevorgang auf einen einzigen Indexzugriff ab. Im Folgenden wird die dafür benötigte Modellierung der im Index abgelegten L-Documents sowie die dazugehörige Anfragemodifikation vorgestellt, deren Eigenschaften diskutiert und gezeigt, dass so ein kontextbasiertes Bewertungsmaß realisiert werden kann. Schließlich wird noch kurz auf die Parameterbelegung bei der Erzeugung des Lucene-Index eingegangen.

Modellierung von L-Documents und Anfragemodifikation Beim Entwurf der Komponenten zur Kontextualisierung und zur Anfragemodifikation stand im Vordergrund, das in Abschnitt 3.2.1.4 eingeführte kombinierte Bewertungsmaß auf die in Abschnitt 4.6.1.3 dargestellte Bewertungsfunktion von Lucence abzubilden. Da, wie dort diskutiert, Lucene selbst nur eine Variante des Vektorraummodells mit $\mathrm{tf.idf}$ implementiert, ging es nicht darum, die exakte Definition der kombinierten Berechnungsfunktion zu realisieren, sondern vielmehr die darin enthaltenen Konzepte auf das Lucene'sche Bewertungsschema zu übertragen. Um den Implementierungsaufwand zu minimieren, wurde dabei das Ziel verfolgt, möglichst die in Lucene bereits implementierte Indexierungs- und Suchtechnologie anzuwenden. Im Folgenden wird eine solche mögliche Übertragung skizziert. Dazu wird zuerst die Lösung vorgestellt und anschließend deren Eigenschaften diskutiert.

Der hier vorgestellte Ansatz zur Übertragung von $\mathrm{ctf.icf}$ auf das Ranking-Modell von Lucene basiert auf der Verwendung der in Abschnitt 4.6.1.1 eingeführten L-Fields. In ihnen werden die Begriffe abgelegt, welche in Knoten mit unterschiedlichem semantischen Abstand vom jeweiligen Dokument auftauchen. Sei $cRange$ ein fester Wert, welcher nach Abschnitt 3.1.3.3 eine obere Schranke für den semantischen Abstand eines Knoten im Kontext eines Dokuments beschreibt. Dann wird für jedes zu indexierende Dokument d ein L-Document d^ℓ erzeugt, welches neben den L-Fields für die URL und den Text des Dokuments die L-Fields $\mathrm{rel}_1, \mathrm{rel}_2, \ldots, \mathrm{rel}_{cRange}$ zur Abspeicherung der Kontextinformation enthält. Ein naheliegender Ansatz wäre nun, die Begriffe aller Knoten, die die semantische Nähe k zu d haben, im L-Field rel_k von d^ℓ abzuspeichern. Dieser Ansatz greift jedoch zu kurz, da die icf-Komponente der Bewertungsfunktion so nicht adäquat abgebildet werden kann. Die im Rahmen dieser Arbeit verfolgte Lösung besteht vielmehr darin, im L-Field rel_k diejenigen Begriffe abzulegen, deren Knoten eine semantische Nähe zu d haben, die *größer oder gleich* k ist. Für das Beispiel sind die entsprechenden Felder in Abbildung 4.5 dargestellt. Dabei sind der Übersichtlichkeit halber im L-Field rel_k diejenigen Begriffe unterstrichen, die semantische Nähe k haben. Auf die Belegung der Parameter Stored, Indexed und Analyzed wird weiter unten eingegangen. Bei der Indexierung wird der Document-Boost (siehe Abschnitt 4.6.1.1) von d^ℓ auf einen Wert gesetzt, der der Wichtigkeit von d entspricht.

Um die auf diese Weise im Index abgelegte Information zur Anfragezeit zur Be-

Begriffe	L-Field	Stored	Indexed	Analyzed
file:///Z:/docs/fb/Doc11.txt	URL	Ja	Nein	Nein
Es taucht wiederholt das Problem auf, dass die mittlere Düse verstopft. Dies führt v.a. links zu unscharfer Klebenaht. Beim Aufbringen zeigt der Kleber …	text	Nein	Ja	Ja
Mittlere Düse wiederholt verstopft. Führt zu unstabiler Klebenaht Dok11	rel_3	Nein	Ja	Ja
Fehlerbericht P23 Hubhebel M18 Mittlere Düse wiederholt verstopft. Führt zu unstabiler Klebenaht Dok11	rel_2	Nein	Ja	Ja
P7 Schwinghebel Teil Maschine Fehlerbericht P23 Hubhebel M18 Mittlere Düse wiederholt verstopft. Führt zu unstabiler Klebenaht Dok11	rel_1	Nein	Ja	Ja

Abbildung 4.5: Beispieldaten bei der Erstellung des Einzelschrittindex.

wertung der Ergebnisse verwenden zu können, muss die vom Benutzer angegebene Anfrage modifiziert werden. Es sei darauf hingewiesen, dass diese Anfragemodifikation allein durch die Strukturierung des Index und die Anforderungen an die Bewertung der Suchresultate motiviert ist. Sie ist sowohl inhaltlich als auch technologisch völlig separiert von der Komponente zur Anfragemodifikation nach Abbildung 4.2, welche eine inhaltliche Umschreibung von Suchanfragen vornehmen kann.

Jeder Begriff der Suchanfrage muss nicht nur im Dokumententext, sondern in allen L-Fields gesucht werden. Um Treffer in solchen L-Fields zu belohnen, welche die Begriffe aus Knoten mit großer semantischer Nähe enthalten, wird zusätzlich vom Term-Boost Gebrauch gemacht. Eine Suchanfrage nach Klebenaht UND Schwinghebel wird somit in folgende Anfrage überführt.

(text:Klebenaht^$textboost$
ODER rel_1:Klebenaht^1
ODER rel_2:Klebenaht^2
...
ODER rel_{cRange}:Klebenaht^$cRange$)
UND (text:Schwinghebel^$textboost$
ODER rel_1:Schwinghebel^1
ODER rel_2:Schwinghebel^2
...
ODER rel_{cRange}:Schwinghebel^$cRange$)

Wie in Abschnitt 4.6.1.3 zur Bewertung von Suchresultaten in Lucene eingeführt, werden die verschiedenen Teile des modifizierten Suchausdrucks mithilfe der Term-Boost-Funktionalität unterschiedlich gewichtet. In der modifizierten Anfrage wird dies in der Lucene-eigenen Schreibweise mittels des ^-Zeichens angezeigt. Entsprechend der oben eingeführten Formel zur Berechnung der Resultatbewertungen steht die Zahl 1 dabei für die Standard-Gewichtung, ein höherer Zahlenwert zeigt eine höhere Gewichtung an. Die Gewichtung des L-Fields text mit der Variable $textboost$ wird weiter unten besprochen.

Für die formale Definition der Umformulierung sei $q = q_1\ b_1\ q_2\ b_2 \ldots b_{k-1}\ q_k$

eine Anfrage aus k Begriffen $q_1, \ldots q_k$, die mit den Operatoren $b_1, \ldots, b_{k-1} \in \{$UND, ODER$\}$ verknüpft sind. Dann wird jeder Begriff q_i zu einem Teilausdruck $q'_i = (\text{text:}q_i\hat{}textboost\ \text{ODER}_{j=1\ldots cRange}\ \text{rel}_j\text{:}q_i\hat{}j)$ erweitert, wobei der ODER-Operator mit Subskript als Kurzform eines iterierten ODER-Ausdrucks, ODER ... ODER ... ODER, zu lesen ist. Die modifizierte Anfrage q' ergibt sich dann aus den modifizierten Begriffen als $q' = q'_1\ b_1\ q'_2\ b_2 \ldots b_{k-1}\ q'_k$.

Diskussion Folgende Merkmale der Modellierung von L-Documents und der Anfragemodifikation erfordern gesonderte Betrachtung.

- Textboost. Über die Variable *textboost* kann zur Anfragezeit das Verhältnis des Einflusses von Treffern im Text bzw. im Kontext von Dokumenten variiert werden. Verglichen mit den Parametern r_1 und r_2 der nach Definition 3.2.4 festgelegten Berechnung kombinierter Begriffsgewichte, entspricht *textboost* dem Verhältnis $\frac{r_1}{r_2}$.

- Kontextgröße. Ebenfalls zur Anfragezeit lässt sich leicht die Größe des in Betracht gezogenen Kontexts einschränken. Um dies zu realisieren, müssen in der modifizierten Suchanfrage lediglich diejenigen L-Fields, die die Begriffe mit kleinerer semantischer Nähe beinhalten, weggelassen werden. Sollte eine Anwendung dies erforderlich machen, so könnte ebenfalls vom Benutzer festgelegt werden, welche der angegebenen Begriffe unbedingt im Text des Dokuments, welche im nahen Kontext und welche im weiteren Kontext vorkommen sollen. Diese Möglichkeit zur Einschränkung der Kontextgröße ist somit eine Verallgemeinerung des in Abschnitt 2.3.1 vorgestellten Find/Near-Ansatzes.

- NICHT-Operator. Die Behandlung des in Abschnitt 3.2.1.5 diskutierten Problems des NICHT-Operators in Suchanfragen wird nach der Methode der adaptiven Kontextgröße behandelt. Dazu muss lediglich die Anzahl der im NICHT-Operator in Betracht gezogenen L-Fields sukzessive verringert werden. Für einen negierten Suchbegriff werden also zunächst neben dem L-Field text alle L-Fields des Kontexts mitbetrachtet. Liefert diese Suche keine Ergebnisse, so wird iterativ jeweils das L-Field mit den Begriffen der kleinsten semantischen Nähe nicht betrachtet. Kommt der negierte Suchbegriff in allen L-Fields der Kontexte aller Dokumente vor, so wird die Negation lediglich auf das text L-Field der Dokumente angewandt.

- Boost. Bei der Ausführung der Anfrage wurde die Gewichtung von Begriffen in unterschiedlicher semantischer Nähe zum Dokument durch die Verwendung der Term-Boost-Funktionalität realisiert. Dieselbe Bewertung ließe sich ebenfalls mit Hilfe des Field-Boosts erzeugen, da die jeweiligen Boost-Faktoren bereits zur Indexierungszeit bekannt sind. Bei der Implementierung des Prototyps wurde die Realisierung mittels Term-Boost vorgezogen, da die eingesetz-

te Version von Lucene[10] bei der Behandlung des Field-Boosts nicht fehlerfrei arbeitete.

Implementierung der Bewertungsmaße Entsprechend der Modellierung der L-Documents und der Anfragemodifikation ergibt sich für die Bewertung der Relevanz eines Dokuments d bzgl. einer modifizierten Anfrage q folgende Berechnung.

Nach der Definition der Bewertungsfunktion in Lucene (siehe Gleichung 4.1):

$$\mathrm{rel}(d,q) = \sum_{t \in q} \left(\mathrm{dNorm}(q,d,t) \cdot \sum_{f \in d} \mathrm{qtw}(q,f,t) \cdot \mathrm{dtw}(d,f,t) \right)$$

Da sich d aus den Feldern $\{\mathsf{text}, \mathrm{rel}_1, \ldots, \mathrm{rel}_{cRange}\}$ zusammensetzt:

$$= \sum_{t \in q} \Bigg(\mathrm{dNorm}(q,d,t) \cdot \Bigg(\underbrace{\mathrm{qtw}(q,\mathsf{text},t) \cdot \mathrm{dtw}(d,\mathsf{text},t)}_{\text{Text}} + \underbrace{\sum_{i=1\ldots cRange} \mathrm{qtw}(q,\mathrm{rel}_i,t) \cdot \mathrm{dtw}(d,\mathrm{rel}_i,t)}_{\text{Kontext}} \Bigg) \Bigg) \tag{4.2}$$

Zur Berechnung der Ähnlichkeit zwischen dem Anfrage- und dem Dokumentenvektor wird das in Lucene realiserte modifizierte Skalarprodukt nach Gleichung 4.1 verwendet. Gleichung 4.2 entspricht in ihrer Form bereits der in Definition 3.2.4 auf Seite 96 beschriebenen Formel zur kombinierten Berechnung der Begriffsgewichte aus dem Text, bzw. dem Kontext der Dokumente. Wie bereits diskutiert, können die Gewichtungsparameter r_1 und r_2 aus Definition 3.2.4 leicht durch eine entsprechende Belegung der Variable *textboost* modelliert werden, die den Term-Boost des L-Fields text angibt. Während der Text-Teil in Gleichung 4.2 bereits offensichtlich die gewünschte Form aufweist, bedarf der Kontext-Teil noch der Diskussion. Nach Gleichung 4.1 lässt dieser sich folgendermaßen aufschlüsseln:

$$\begin{aligned} &\sum_{i=1\ldots cRange} \mathrm{qtw}(q,\mathrm{rel}_i,t) \cdot \mathrm{dtw}(d,\mathrm{rel}_i,t) \\ = &\sum_{i=1\ldots cRange} \mathrm{idf}(\mathrm{rel}_i,t) \cdot \mathrm{tb}(\mathrm{rel}_i,t) \cdot \mathrm{qNorm}(q) \\ &\qquad \cdot \mathrm{tf}(\mathrm{rel}_i,t) \cdot \mathrm{idf}(\mathrm{rel}_i,t) \cdot \mathrm{fb}(\mathrm{rel}_i) \cdot \mathrm{db}(d) \cdot \mathrm{fNorm}(\mathrm{rel}_i,d) \end{aligned}$$

[10] Version 1.4.3

Dabei gilt im Protoypen $\mathrm{fb}(f) = 1$ für alle L-Fields und $\mathrm{tb}(\mathrm{rel}_i, t) = i$. Weiterhin wird der Document-Boost $\mathrm{db}(d)$, sowie der qNorm-Faktor aus der Summe herausgezogen. Da letzterer lediglich dazu dient, eine Vergleichbarkeit von Ergebnisbewertungen zwischen verschiedenen Anfragen herzustellen und keinerlei Einfluss auf die Reihenfolge der Ergebnisdokumente einer Anfrage hat, wird auf eine weitere Diskussion hierzu verzichtet. Die so vereinfachte Summe hat dann folgende Form:

$$\sum_{i=1\ldots cRange} \underbrace{\mathrm{idf}(\mathrm{rel}_i), t}_{\approx\, \mathrm{ctf\,icf}(q,t)} \cdot \underbrace{i \cdot \mathrm{tf}(\mathrm{rel}_i, t) \cdot \mathrm{idf}(\mathrm{rel}_i, t) \cdot \mathrm{fNorm}(\mathrm{rel}_i, d)}_{\approx\, \mathrm{ctf\,icf}(d,t)} \tag{4.3}$$

Wie durch die Klammern angedeutet entspricht der erste Teil der Formel der Berechnung von Begriffsgewichten mittels $\mathrm{ctf.icf}$ im Anfragevektor. Analog zur Berechnung der Begriffsgewichte in Lucene nach Gleichung (4.1) enthalten die Begriffsgewichte des Anfragevektors auch hier keinen tf-Anteil.

Die Berechnung der Begriffsgewichte im Dokumentenvektor enthalten hingegen – ebenfalls analog zur Berechnung der Begriffsgewichte in Lucene – sowohl einen tf- als auch einen idf-Anteil sowie den Normierungsfaktor fNorm. Der Faktor i ist, wie oben beschrieben, dafür zuständig, Treffer auf Knoten mit kleinerem Abstand zu belohnen.

Die Umsetzung von $\mathrm{ctf.icf}$ durch tf und idf, wie in Formel 4.3 beschrieben, basiert auf zwei Eigenschaften:

- Summation. Der Summenausdruck im Zähler von ctf nach Definition 3.2.1 wird im Prototyp nicht durch die Berechnung der Begriffsgewichte geleistet, sondern durch das um das Konzept der L-Fields erweiterte Vektorraummodell. Dies wird durch die Modellierung der L-Documents ermöglicht, bei der jeweils Begriffe mit unterschiedlicher semantischer Nähe in unterschiedlichen L-Fields abgelegt werden.
  Für die Berechnung von icf zieht diese Verschiebung der Summation keine nennenswerten Änderungen nach sich.

- Inhalt der L-Fields. Oben wurde beschrieben, dass in L-Field rel_i nicht nur die Begriffe enthalten sind, welche vom Dokument aus in der semantischen Nähe i zu finden sind, sondern auch noch alle Begriffe, welche näher am Dokument liegen. Diese Modellierung ermöglicht eine bessere Umsetzung von icf nach Definition 3.2.2. Auf diese Weise geht nicht nur die Anzahl des Auftauchens eines Begriffs in der jeweiligen semantischen Nähe in die Berechnung mit ein, sondern zusätzlich die Anzahl des Auftauchens des Begriffs in größerer semantischer Nähe.
  Für die Berechnung von ctf bewirkt diese Modellierung, dass Begriffe, die in größerer semantischer Nähe gefunden werden, mehrfach in die Bewertung

miteingehen. Dies ist ein durchaus erwünschter Effekt, da so de facto lediglich der Term-Boost von Begriffen in großer Nähe leicht verstärkt wird.

- Knoten vs. Begriffe. Bei genauer Betrachtung leistet der Einzelschrittindex sogar etwas mehr als durch die Bewertungsfunktion $\mathrm{ctf.icf}$ festgelegt. Während in den Definitionen 3.2.1 und 3.2.2 jeweils die Anzahl relevanter Knoten im Kontext eines Dokuments in die Berechnung eingehen, wird beim Einzelschrittindex sogar die Anzahl der relevanten Begriffe in den Knoten beachtet. Während $\mathrm{ctf.icf}$ alle Knoten gleich behandelt, unabhängig von der Anzahl der darin enthaltenen relevanten Begriffe, nimmt der Einzelschrittindex hier eine Differenzierung vor. Wäre diese Differenzierung unerwüscht, so ließe sie sich leicht dadurch beheben, dass vor der Aufnahme in den Index alle Mehrfachnennungen von Begriffen in Knotenbeschriftungen entfernt werden.

Durch das Zusammenspiel der Modellierung der L-Documents, der oben beschriebenen Anfragemodifikation sowie der bereits in Lucene implementierten Technologie zur Bewertung von Suchresultaten kann somit eine Funktion zur Bewertung von Suchresultaten realisiert werden, welche in ihrem Aufbau der in Abschnitt 3.2.1.4 vorgestellten kombinierten Bewertungsfunktion entspricht.

Indexerzeugung Die Komponenten zur Indexerzeugung und zum Zugriff auf den Index sind entsprechend den obigen Ausführungen durch die entsprechenden Lucene-Komponenten realisiert. Gemäß Abbildung 4.5 werden für die L-Fields für den Text sowie für den Kontext der Dokumente die Parameter folgendermaßen gesetzt:

- Stored: Nein. Weder Text noch Kontext können zur Anfragezeit aus dem Kontext ausgelesen werden. Dies verkleinert die Indexgröße, hat jedoch zur Folge, dass die Anzeige der Suchergebnisse keine aus dem Index gespeiste Dokumenten- oder Kontextvorschau enthalten kann.
- Indexed: Ja. Nach allen Begriffen im Text oder im Kontext eines Dokuments kann gesucht werden.
- Analyzed: Die im Prototypen verwendete Komponente zur Textanalyse[11] ersetzt alle Großbuchstaben durch Kleinbuchstaben und entfernt Stoppworte.

Für das L-Field URL wird Stored auf Ja gesetzt, was bewirkt, dass zur Anfragezeit beim Zugriff auf den Index die URL jedes Dokuments gelesen werden kann. Da nach URLs jedoch nicht gesucht werden können soll, ist der Parameter Indexed auf Nein gesetzt; da URLs keinerlei Analyse bedürfen, ist Analyzed ebenfalls negiert.

4.6.3 Doppelschrittindex für Schlüsselwortsuche

Die Modellierung der Strukturen des Einzelschrittindex enthält eine gewisse Redundanz, da für Knoten des ContextGraphs, die viele Begriffe beinhalten, im invertierten

[11] Die entsprechende Lucene-Klasse heißt: `org.apache.lucene.analysis.standard.StandardAnalyzer`

Index für jeden dieser Begriffe die gleiche Menge an Dokumenten gespeichert werden muss. Der in diesem Abschnitt vorgestellte Doppelschrittindex vermeidet diese Redundanz.

Um diese Redundanz zu verdeutlichen sei folgendes Beispiel gegeben: Ein Knoten mit Begriffen `A` `B` und `C` trete im Kontext zweier Dokumente d_1 und d_2 auf. Im Einzelschrittindex wird zu jedem der drei Begriffe ein Verweis auf beide Dokumente abgelegt. Diese Modellierung ist insofern redundant, da stets für alle Begriffe eines Knotens Verweise auf dieselben Dokumente abgespeichert werden. Ein weiteres, ausführlicheres Beispiel wird weiter unten gegeben.

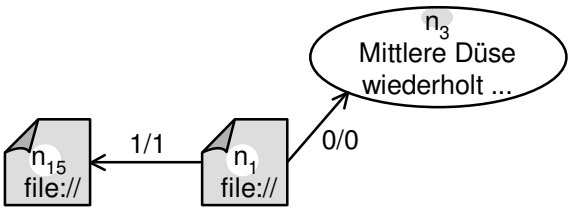
Modellierung von L-Documents und Anfragemodifikation Die Idee des Doppelschrittindex liegt darin, alle Begriffe eines Knoten im ContextGraph zusammenzufassen und so die dem Einzelschrittindex inhärente Redundanz zu vermeiden. Die Umsetzung dieser Idee macht die Verwendung von zwei separaten Indexstrukturen erforderlich:

- Knotenindex. Der Knotenindex realisiert eine effiziente Zugriffstruktur, welche einen im ContextGraph enthaltenen Begriff auf die Menge der Knoten abbildet, welche diesen Begriff enthalten. Der Name "Knotenindex" ist motiviert durch die Tatsache, dass das Ergebnis eines Zugriffs auf diesen Index aus einer Menge von Knoten, genauer Knotenidentifikatoren, besteht.

- Dokumentindex. Der Dokumentindex bildet jeden im ContextGraph enthaltenen Knoten auf die Menge der Dokumente ab, in deren Kontext er vorkommt. Analog zum Einzelschrittindex wird zusätzlich mittels L-Fields der semantische Abstand zwischen Dokument und Knoten notiert. Da der Text eines Dokuments jeweils nur für das Dokument selbst relevant ist, werden die Begriffe des Texts ebenfalls in den Dokumentindex aufgenommen. Analog zum Knotenindex soll durch den Namen "Dokumentindex" ausgedrückt werden, dass das Ergebnis eines Zugriffs aus einer Menge von Dokumenten, bzw. deren Identifikatoren, besteht.

Zur Illustration des Unterschieds zwischen Einzelschritt- und Doppelschrittindex sei der ContextGraph des Standardbeispiels so modifiziert, wie in Abbildung 4.6 (a) dargestellt. Das bereits bekannte Dokument im Knoten n_1 mit der bereits bekannten Kurzbeschreibung referenziert nun ein weiteres Dokument. Diese Referenz wird im ContextGraph durch eine Kante der Länge 1 modelliert.

Abbildung 4.6 (b) zeigt die logische Modellierung des Einzelschrittindex. Für jeden Begriff wird jedes L-Field jedes Dokuments einzeln aufgelistet. Der Dokumentenidentifikator kann dabei vom System vorgegeben werden. Für die Beispiele in diesem Abschnitt wird jeweils der Identifikator des entsprechenden Dokumentenknoten im ContextGraph gewählt.



(a)

Begriff	L-Field	Doc-ID
mittlere	rel_1	n_1
mittlere	rel_2	n_1
mittlere	rel_3	n_1
düse	rel_1	n_1
düse	rel_2	n_1
düse	rel_3	n_1
wiederholt	rel_1	n_1
wiederholt	rel_2	n_1
wiederholt	rel_3	n_1
file:///Z:/...	URL	n_1
taucht	text	n_1
mittlere	rel_1	n_{15}
mittlere	rel_2	n_{15}
düse	rel_1	n_{15}
düse	rel_2	n_{15}
wiederholt	rel_1	n_{15}
wiederholt	rel_2	n_{15}
file:///Z:/...	URL	n_{15}

(b)

Knotenindex

Begriff	L-Field	Doc-ID
mittlere	std	n_3
düse	std	n_3
wiederholt	std	n_3

Dokumentindex

Begriff	L-Field	Doc-ID
n_3	rel_1	n_1
n_3	rel_2	n_1
n_3	rel_3	n_1
file:///Z:/...	URL	n_1
taucht	text	n_1
n_3	rel_1	n_{15}
n_3	rel_2	n_{15}
file:///Z:/...	URL	n_{15}

(c)

Abbildung 4.6: Einzelschrittindex vs. Doppelschrittindex. (a) Modifizierter Beispielgraph (b) Logischer Aufbau des Einzelschrittindex. (c) Logischer Aufbau des Doppelschrittindex bestehend aus Knotenindex und Dokumentindex.

Die logische Modellierung des Doppelschrittindex ist in Abbildung 4.6 (c) abgebildet. Da der Knotenindex lediglich die Abbildung von Begriffen auf Knoten vornimmt, wird nur ein einziges L-Field benötigt, welches in der Abbildung mit std benannt ist. Der Knotenindex ist ein vollkommen eingeständiger Lucene-Index, welcher durch die Komponente zur Indexerzeugung erstellt wird. Die L-Documents dieses Index werden genau durch die beschrifteten Knoten des ContextGraphs gebildet. Die Tatsache, dass die rechte Spalte des Knotenindex mit Doc-ID überschrieben ist, soll darauf hinweisen, dass hier die Identifikatoren der L-Documents aufgeführt werden.

Der Dokumentindex bildet die Knoten des ContextGraphs auf für sie relevante Dokumente ab. Im Beispiel wird der einzige beschriftete Knoten n_3 den Dokumenten in Knoten n_{15} und n_1 zugeordnet. Analog zu den Überlegungen zur Implementierung der Bewertungsmaße in Abschnitt 4.6.2 wird n_3 drei L-Fields des Dokuments in Knoten n_1 und zwei L-Fields des Dokuments in Knoten n_{15} zugeordnet. Zusätzlich muss – wie beim Einzelschrittindex – zu jedem Dokument die entsprechende URL und der Text des Dokuments (im Beispiel durch den Begriff taucht symbolisiert) abgelegt werden.

Das Abspeichern des Dokumententexts im Dokumentenindex hat gegenüber der Abspeicherung im Knotenindex den Vorteil, dass die Bewertung der Ergebnisse mit-

tels tf.idf auf natürliche Weise auf Basis des Dokumentenindex durchgeführt und somit direkt in die kombinierte Resultatsbewertung einfließen kann.

Da der Doppelschrittindex aus zwei autonomen Indexen besteht, muss die Anfrage ebenfalls aus zwei Teilen bestehen. Im ersten Teil wird dabei für jeden Suchbegriff die Menge der Knoten bestimmt, in denen der Begriff auftaucht, im zweiten wird dann daraus die relevante Dokumentenmenge generiert. Für das obige Beispiel der Anfrage nach Klebenaht ODER Schwinghebel werden zwei Anfragen an den Knotenindex gestellt: std:Klebenaht und std:Schwinghebel. Das Ergebnis der beiden Anfragen besteht aus einer Menge $\{n_1, \ldots, n_k\}$ an Knoten, die den Begriff Klebenaht und einer Menge $\{m_1, \ldots, m_\ell\}$, die den Begriff Schwinghebel enthalten.

Im zweiten Schritt wird dann aus diesen Mengen eine Anfrage an den Dokumentenindex gestellt, die analog zur Anfragemodifikation beim Einzelschrittindex verschiedene Term-Boosts zur Modellierung unterschiedlicher semantischer Abstände einsetzt. Die modifizierte Anfrage für das Beispiel lautet dann

$$
\begin{array}{lll}
 & (& (\mathsf{rel}_1{:}\mathsf{n}_1\hat{}1 \text{ ODER } \mathsf{rel}_2{:}\mathsf{n}_1\hat{}2 \text{ ODER } \ldots \mathsf{rel}_{cRange}{:}\mathsf{n}_1\hat{}cRange) \\
 & \text{ODER} & (\mathsf{rel}_1{:}\mathsf{n}_2\hat{}1 \text{ ODER } \mathsf{rel}_2{:}\mathsf{n}_2\hat{}2 \text{ ODER } \ldots \mathsf{rel}_{cRange}{:}\mathsf{n}_2\hat{}cRange) \\
 & \text{ODER} & \ldots \\
 & \text{ODER} & (\mathsf{rel}_1{:}\mathsf{n}_k\hat{}1 \text{ ODER } \mathsf{rel}_2{:}\mathsf{n}_k\hat{}2 \text{ ODER } \ldots \mathsf{rel}_{cRange}{:}\mathsf{n}_k\hat{}cRange) \\
 & \text{ODER} & (\mathsf{text}{:}\mathsf{Klebenaht}) \\
 &) & \\
\text{ODER} & (& (\mathsf{rel}_1{:}\mathsf{m}_1\hat{}1 \text{ ODER } \mathsf{rel}_2{:}\mathsf{m}_1\hat{}2 \text{ ODER } \ldots \mathsf{rel}_{cRange}{:}\mathsf{m}_1\hat{}cRange) \\
 & \text{ODER} & (\mathsf{rel}_1{:}\mathsf{m}_2\hat{}1 \text{ ODER } \mathsf{rel}_2{:}\mathsf{m}_2\hat{}2 \text{ ODER } \ldots \mathsf{rel}_{cRange}{:}\mathsf{m}_2\hat{}cRange) \\
 & \text{ODER} & \ldots \\
 & \text{ODER} & (\mathsf{rel}_1{:}\mathsf{m}_\ell\hat{}1 \text{ ODER } \mathsf{rel}_2{:}\mathsf{m}_\ell\hat{}2 \text{ ODER } \ldots \mathsf{rel}_{cRange}{:}\mathsf{m}_\ell\hat{}cRange) \\
 & \text{ODER} & (\mathsf{text}{:}\mathsf{Schwinghebel}) \\
 &). &
\end{array}
$$

D.h. für jeden gefundenen Knoten wird eine Teilanfrage generiert, welche in allen L-Fields $\mathsf{rel}_1, \ldots, \mathsf{rel}_{cRange}$ des Kontexts sucht. Gleichzeitig wird ebenfalls auf dem L-Field text gesucht, in dem die Texte der Dokumente abgelegt sind.

Diskussion Der in diesem Abschnitt eingeführte Doppelschrittindex hat gegenüber dem Einzelschrittindex sowohl Vorteile als auch Nachteile:

Der Vorteil des Doppelschrittindex liegt in der Vermeidung von im Index redundant abgelegter Information. Eine kontextbasierte Suchmaschine benötigt, verglichen mit herkömmlichen Suchmaschinen, aufgrund der beim Suchprozess zusätzlich verwendeten Information, inhärent größere Indexe und daher mehr Plattenplatz. Die Verwendung des Doppelschrittindex kann helfen, diesen Mehraufwand an Plattenplatz zu reduzieren. Diese Reduktion der Indexgröße wirkt sich vor allem in Szenarien mit folgenden Charakteristika aus:

- Große Knoten. Wird einem beliebigen Knoten ein zusätzlicher Begriff hinzugefügt, so führt dies im Einzelschrittindex zu einer zusätzlichen Zeile für jedes Kontext-L-Field jedes Dokuments, in dessen Kontext der Knoten zu finden ist.

Im Doppelschrittindex hingegen wird dem Knotenindex genau ein weiterer Eintrag hinzugefügt.

- Große Kontextüberlappung. Kommen viele Knoten im Kontext mehrerer Dokumente vor, so werden die Begriffe dieser Knoten im Einzelschrittindex für jedes Dokument einzeln aufgelistet. Im Doppelschrittindex hingegen muss im Dokumentenindex lediglich der Knotenidentifikator mit entsprechenden L-Fields eingetragen werden.

Die Vorteile bzgl. der Einsparung von Redundanz beim Abspeichern des Index implizieren jedoch den folgenden Nachteil.

- Anfrageperformanz. Die Ausführungsgeschwindigkeit von Suchanfragen hat unter der Tatsache zu leiden, dass, nicht wie beim Einzelschrittindex, pro Anfrage ein Index-Zugriff durchgeführt werden muss, sondern mehrere Anfragen nötig werden. Dies wirkt um so schwerer, wenn die Art der generierten Anfragen in Betracht gezogen wird. Während sich die modifizierte Anfrage beim Einzelschrittindex um den Faktor $cRange$ verlängert, wird die Länge der Anfrage auf dem Dokumentenindex zusätzlich durch die Anzahl der im ersten Schritt gefundenen Knoten bestimmt. Diese Anzahl liegt im Experiment leicht bei mehreren hundert oder tausend Knoten, was eine entsprechende Zunahme der Anfragelänge bedeutet. Da bei Lucene die Ausführungsgeschwindigkeit einer Anfrage teilweise von der Länge der Anfrage abhängt, impliziert diese Eigenschaft eine Verminderung der Anfrageperformanz.

Indexerzeugung Die Belegung der Indexierungsparameter des Doppelschrittindex ist in Abbildung 4.7 dargestellt. Das einzige L-Field des Knotenindex std wird lediglich dazu verwendet, die Abbildung von Begriffen auf Knotenidentifikatoren zu realisieren. Sein Inhalt wird daher analysiert und indexiert, aber nicht im Index gespeichert. Das L-Field text des Dokumentindex beinhaltet den Text der indexierten Dokumente. Es wird daher ebenfalls analysiert und indexiert, jedoch nicht im Index gespeichert. Die L-Fields der Kontext-Information rel_x für $x = 1...cRange$ beinhalten für jedes Dokument die Knotenidentifikatoren aus dessen Kontext. Diese werden ebenfalls indexiert und nicht im Index gespeichert. Da in diesen L-Fields jedoch ausschließlich Knotenidentifikatoren abgespeichert werden, ist eine Analyse überflüssig. Das L-Field für URLs im Dokumentindex schließlich hat dieselben Charakteristika wie im Einzelschrittindex. Wieder muss lediglich zu jedem Dokument die entsprechende URL im Index abgespeichert werden.

4.6.4 Doppelschrittindex für Ähnlichkeitssuche

Die Ähnlichkeitssuche und das damit verbundene Bewertungsmaß $\mathrm{nr.inf}$ wurde in Abschnitt 3.2.3 eingeführt. Es basiert auf der Idee, dass zwei Dokumente als ähn-

Index	L-Field	Stored	Indexed	Analyzed
Knotenindex	std	Nein	Ja	Ja
Dokumentindex	text	Nein	Ja	Ja
Dokumentindex	rel_x	Nein	Ja	Nein
Dokumentindex	URL	Ja	Nein	Nein

Abbildung 4.7: Parameterbelegung beim Doppelschrittindex.

lich angesehen werden, falls ihre Kontexte überlappen. In diesem Abschnitt wird eine einfache Lösung zur Realisierung der Ähnlichkeitssuche vorgestellt, welche durch eine minimale Modifikation aus dem oben eingeführten Doppelschrittindex für Schlüsselwortsuche hervorgeht.

Modellierung von L-Documents und Anfragemodifikation Wie bereits in Abschnitt 3.2.3 motiviert, bewertet die kontextbasierte Ähnlichkeitssuche mittels $\mathrm{nr.inf}$ zwei Dokumente genau dann als ähnlich, wenn die Schnittmenge ihrer Kontexte groß ist. Dabei haben solche Knoten mehr Gewicht, die möglichst große semantische Nähe zu möglichst beiden Dokumenten haben.

Die Suche nach ähnlichen Dokumenten kann daher in zwei Schritte unterteilt werden. Im folgenden werde nach ähnlichen Dokumenten zu Dokument d gesucht.

- Bestimmung des Kontexts. Für d müssen alle Knoten in $\mathrm{context}(d)$ bestimmt werden sowie die jeweilige semantische Nähe zu d.
- Bestimmung ähnlicher Dokumente. Basierend auf den Ergebnissen des ersten Schritts, werden Dokumente bestimmt, welche zu den Knoten aus $\mathrm{context}(d)$ die größte semantische Nähe haben.

Im Vergleich mit dem Doppelschrittindex zur Schlüsselwortsuche unterscheidet sich das Verfahren lediglich im ersten Schritt. In beiden Fällen besteht das Ergebnis des ersten Schritts aus einer Menge von Knoten, die im zweiten Schritt zur kontextbasierten Suche verwendet werden. Im Fall der Schlüsselwortsuche wird das Ergebnis des ersten Schritts mittels einer Textsuche auf den Knoten des ContextGraphs bestimmt. Bei der Ähnlichkeitssuche hingegen wird sie direkt aus dem Kontext des vorgegebenen Dokuments erzeugt.

Da der zweite Schritt in beiden Fällen auf derselben Information beruht, kann somit für die Ähnlichkeitssuche der Dokumentindex der Schlüsselwortsuche verwendet werden.

Die Realisierung des ersten Schritts der Ähnlichkeitssuche erfordert, dass jedes Dokument auf die Menge aller Knoten in seinem Kontext abgebildet werden kann. Diese Aufgabe lässt sich bei minimalem Aufwand mit einer leichten Modifikation der Indexstruktur des Doppelschrittindex für Schlüsselwortsuche realisieren. Setzt man den Stored-Parameter der L-Fields $\mathsf{rel}_1, \ldots, \mathsf{rel}_{cRange}$ des Dokumentindex auf Ja, so lässt sich nach Abschnitt 4.6.1 zu jedem Dokument d die Menge der Identifikatoren aller Knoten aus $\mathrm{context}(d)$ berechnen.

Die Anfrage im ersten Schritt ist somit keine Schlüsselwortsuche, sondern ein Abruf der im Index zu einem gegebenen Dokument abgelegten Information. Das Ergebnis besteht aus einer Menge von Knoten, die mit ihrer jeweiligen semantischen Nähe annotiert sind. Für das Beispieldokument in Knoten n_1 aus Abbildung 4.3 ergibt sich also etwa folgendes Ergebnis:

rel_3: n_2, n_3
rel_2: n_4, n_6, n_7, n_{10}, n_{11}, n_{14} (, n_2, n_3)
rel_1: n_5, n_8, n_9, n_{12}, n_{13} (, n_4, n_6, n_7, n_{10}, n_{11}, n_{14}, n_2, n_3)

Dabei sind die Knoten in Klammern geschrieben, welche durch die oben beschriebene Eigenschaft des Dokumentindex hinzukommen, bei dem L-Fields, die Knoten kleinerer semantischer Nähe beinhalten, auch stets alle Knoten größerer semantischer Nähe beinhalten. Diese Eigenschaft ist lediglich durch das Layout des Dokumentindex gegeben und für die weitere Betrachtung irrelevant; zur weiteren Berechnung werden diese Knotenduplikate ausgefiltert.

Im zweiten Schritt wird nun analog zum zweiten Schritt des Doppelschrittindex zur Schlüsselwortsuche eine Anfrage generiert, welche – entsprechend der Bewertungsfunktion $\mathrm{nr.inf}$ – solche Dokumente belohnt, die möglichst nahe zu möglichst vielen Knoten sind, denen das gegebene Dokument möglichst nahe ist.

Um die kontextbasierte Ähnlichkeit zweier Dokumente zu bewerten, wurde in Abschnitt 3.2.3 vorgeschlagen, im nr-Teil der Bewertungsfunktion die semantische Nähe der einzelnen Knoten zu beiden Dokumenten auszuwerten. Da durch die Verwendung des Skalarprodukts auf diese Weise die jeweiligen Werte der semantischen Nähe multipliziert werden, wird in der Umsetzung im vorliegenden Prototyp ebenfalls das Produkt der jeweiligen Werte der semantischen Nähe gebildet und als Term-Boost in der Anfrage berücksichtigt. Ein Knoten n mit $\mathrm{sim}_{cRange}(d, n) = k$ wird also aufgrund seiner Nähe zu d mit Field-Boost k versehen. Die daraus zu generierende Teilanfrage lautet demnach rel_1:n^(k·1) ODER rel_2:n^(k·2) ODER ... ODER rel_{cRange}:n^(k·$cRange$). Für das Beispiel ergibt sich demnach die Anfrage:

rel_1:n_2 ^3	ODER rel_2:n_2 ^6	ODER rel_3:n_2 ^9	$\mathrm{sim}(d, n) = 3$
ODER rel_1:n_3 ^3	ODER rel_2:n_3 ^6	ODER rel_3:n_3 ^9	
ODER rel_1:n_4 ^2	ODER rel_2:n_4 ^4	ODER rel_3:n_4 ^6	$\mathrm{sim}(d, n) = 2$
ODER rel_1:n_6 ^2	ODER rel_2:n_6 ^4	ODER rel_3:n_6 ^6	
ODER rel_1:n_7 ^2	ODER rel_2:n_7 ^4	ODER rel_3:n_7 ^6	
ODER rel_1:n_{10}^2	ODER rel_2:n_{10}^4	ODER rel_3:n_{10}^6	
ODER rel_1:n_{11}^2	ODER rel_2:n_{11}^4	ODER rel_3:n_{11}^6	
ODER rel_1:n_{14}^2	ODER rel_2:n_{14}^4	ODER rel_3:n_{14}^6	
ODER rel_1:n_5 ^1	ODER rel_2:n_5 ^2	ODER rel_3:n_5 ^3	$\mathrm{sim}(d, n) = 1$
ODER rel_1:n_8 ^1	ODER rel_2:n_8 ^2	ODER rel_3:n_8 ^3	
ODER rel_1:n_9 ^1	ODER rel_2:n_9 ^2	ODER rel_3:n_9 ^3	
ODER rel_1:n_{12}^1	ODER rel_2:n_{12}^2	ODER rel_3:n_{12}^3	
ODER rel_1:n_{13}^1	ODER rel_2:n_{13}^2	ODER rel_3:n_{13}^3	

Implementierung der Bewertungsmaße Die im letzten Abschnitt vorgestellte Anfragemodifikation für den zweiten Schritt des Doppelschrittindex zur Ähnlichkeitssuche erfüllt die in Abschnitt 3.2.3 aufgestellten Anforderungen an eine Funktion zur Bewertung der kontextbasierten Ähnlichkeit von Dokumenten.

Analog zum Einzelschrittindex (siehe Formel (4.3) in Abschnitt 4.6.2) kann der Beitrag eines Knoten n für die kontextbasierte Ähnlichkeit eines Dokuments d^* zu einem gegebenen Dokument d durch folgende Formel beschrieben werden, dabei sei $\mathrm{sim}_{cRange}(d, n) = j$ und $\mathrm{sim}_{cRange}(d^*, n) = i$:

$$\sum_{i=1\ldots cRange} \mathrm{idf}(\mathrm{rel}_i, n) \cdot j \cdot i \cdot \mathrm{tf}(\mathrm{rel}_i, n) \cdot \mathrm{idf}(\mathrm{rel}_i, n) \cdot \mathrm{fNorm}(\mathrm{rel}_i, d)$$

Da $\mathrm{tf}(\mathrm{rel}_i, n) = 1$ ergibt sich

$$= \sum_{i=1\ldots cRange} \underbrace{\mathrm{idf}(\mathrm{rel}_i, n) \cdot j}_{\approx\, \mathrm{nrinf}(d,n)} \cdot \underbrace{i \cdot \mathrm{idf}(\mathrm{rel}_i, n) \cdot \mathrm{fNorm}(\mathrm{rel}_i, d)}_{\approx\, \mathrm{nrinf}(d^*,n)}. \tag{4.4}$$

Diese Formel enthält die in Abschnitt 3.2.3 beschriebenen Merkmale der Bewertungsfunktion $\mathrm{nr.inf}$. Lediglich bei der Berechnung von $\mathrm{inf}(d, n)$ ergibt sich die Abweichung, dass genau genommen die Funktion $\mathrm{inf}(d^*, n)$ berechnet wird.

Die Berechnung der aus Text- und Kontextinformation kombinierten Begriffsgewichte nach Definition 3.2.6 in Abschnitt 3.2.3 erfolgt analog zur Schlüsselwortsuche. Dazu wird die Anfrage um einen Teil erweitert, der im L-Field text des Dokumentenindex nach Begriffen sucht, die dem Text des ursprünglich gegebenen Dokuments entnommen sind.

4.7 Semantische Suche

Wird vom Benutzer eine Anfrage an die Suchmaschine abgesetzt, so wird diese über den Webserver an eine Komponente zur semantischen Suche geleitet, die die Anfrage ihrerseits an die Komponente zur Indexauswertung weitergibt.

In Abschnitt 2.5 wurde ein Anzahl an Vorschlägen zur semantischen Suche vorgestellt. Einige dieser Ansätze, z.B. TAP und ISRA, beruhen auf ontologiegestützter Anfragemodifikation und Resultatfilterung. Dazu wird ggf. die Anfrage eines Benutzers in mehrere Index-Abfragen umgewandelt. Da der Fokus der vorliegenden Arbeit nicht auf der Einbindung von Ontologien zur Anfragezeit liegt, wurde für den Prototypen eine einfache Variante implementiert, die sowohl die Anfragen des Benutzers, als auch das Ergebnis der Indexauswertung unverändert weiterreicht. Diese Komponente kann in Zukunft leicht durch andere Implementierungen etwa zur Realisierung einer ontologiegestützten Anfragemodifikation und Resultatfilterung ersetzt werden.

4.8 Benutzerschnittstelle

Die Benutzerschnittstelle der Suchmaschine wurde im Prototyp über ein Web-Interface realisert, da so bei minimalen Ressourcenanforderungen eine maximale Verfügbarkeit gewährleistet werden kann. Die Webseite der Suchresultate wird mit Java-Servlets der Java 2 Enterprise Edition (J2EE) generiert. Als Servlet-Container kommt die Tomcat[12]-Implementierung der Apache Software Foundation zum Einsatz. Um dem Benutzer die Struktur des ContextGraphs leichter verständlich zu machen, wurde basierend auf der Touchgraph[13]-Bibliothek eine interaktive Darstellung des ContextGraphs enwickelt. Diese Darstellung ermöglicht es dem Benutzer, per Mausklick im ContextGraph zu navigieren. Je nach Bedarf können Teilgraphen expandiert und kollabiert werden. Um die Übersichtlichkeit zu verbessern, wurde ein Kantenfilter implementiert, über den sich innerhalb eines angezeigten Teilgraphen alle Kanten eines Typs gleichzeitig ein- und ausblenden lassen.

Die Benutzeroberfläche zur Suche ist in Abbildung 4.8 dargestellt. Unterhalb des Eingabefelds für Suchanfragen (a) befindet sich das Feld zur Parametereingabe (b). Hier können vom Benutzer die folgenden Parameter eingestellt werden:

- Number of Results. Die Anzahl der angezeigten Resultate.
- ContextRange. Die Belegung des Parameters $cRange$ im Bereich von 0 bis zu dem Wert, der bei der Berechnung der kürzesten Wege verwendet wurde.
- Not-Range. Die manuelle Belegung des Parameters $nRange$ im gleichen Bereich wie $cRange$.
- Content Influence. Die Gewichtung von textbasierter vs. kontextbasierter Bewertung nach Definition 3.2.4 bzw. die Belegung der Variablen $textboost$ in den Abschnitten 4.6.2 und 4.6.3.

Unterhalb des Parameterfelds wird die Liste der Ergebnisdokumente (c) angezeigt. Sie ist in drei Spalten aufgeteilt. Die Färbung der Quadrate in der linken Spalte weist auf die Bewertung des Dokuments hin. Je besser ein Dokument bewertet wird, desto dunkler ist die Farbe des jeweiligen Quadrats. Die mittlere Spalte enthält die Dateinamen der Dokumente, die rechte besteht aus einer Schaltfläche, die bewirkt, dass das jeweilige Dokument in den Fokus des interaktiven ContextGraph-Browsers (d) geladen wird. Die Werkzeugleiste am unteren Ende des ContextGraph-Browsers dient dazu, die Navigation zu erleichtern und bietet Möglichkeiten zur Skalierung und Drehung des Graphen sowie zur Anwendung eines Fischaugeneffekts, der den dargestellten Graphen auf eine Kugeloberfläche projiziert und somit die Übersichtlichkeit verbessert.

Zur Bewertung der Suchresultate ist der Benutzer nicht ausschließlich auf die Einfärbung des oben besprochenen Quadrats angewiesen. Vielmehr besteht die Möglichkeit, die Zusammensetzung der Bewertung nachzuvollziehen. Ein Klick auf das Quadrat öffnet hierfür ein Fenster, welches die Berechnung der Bewertung erklärt.

[12] `http://tomcat.apache.org/` [29.05.2006]
[13] `http://www.touchgraph.com/`[29.11.2006]

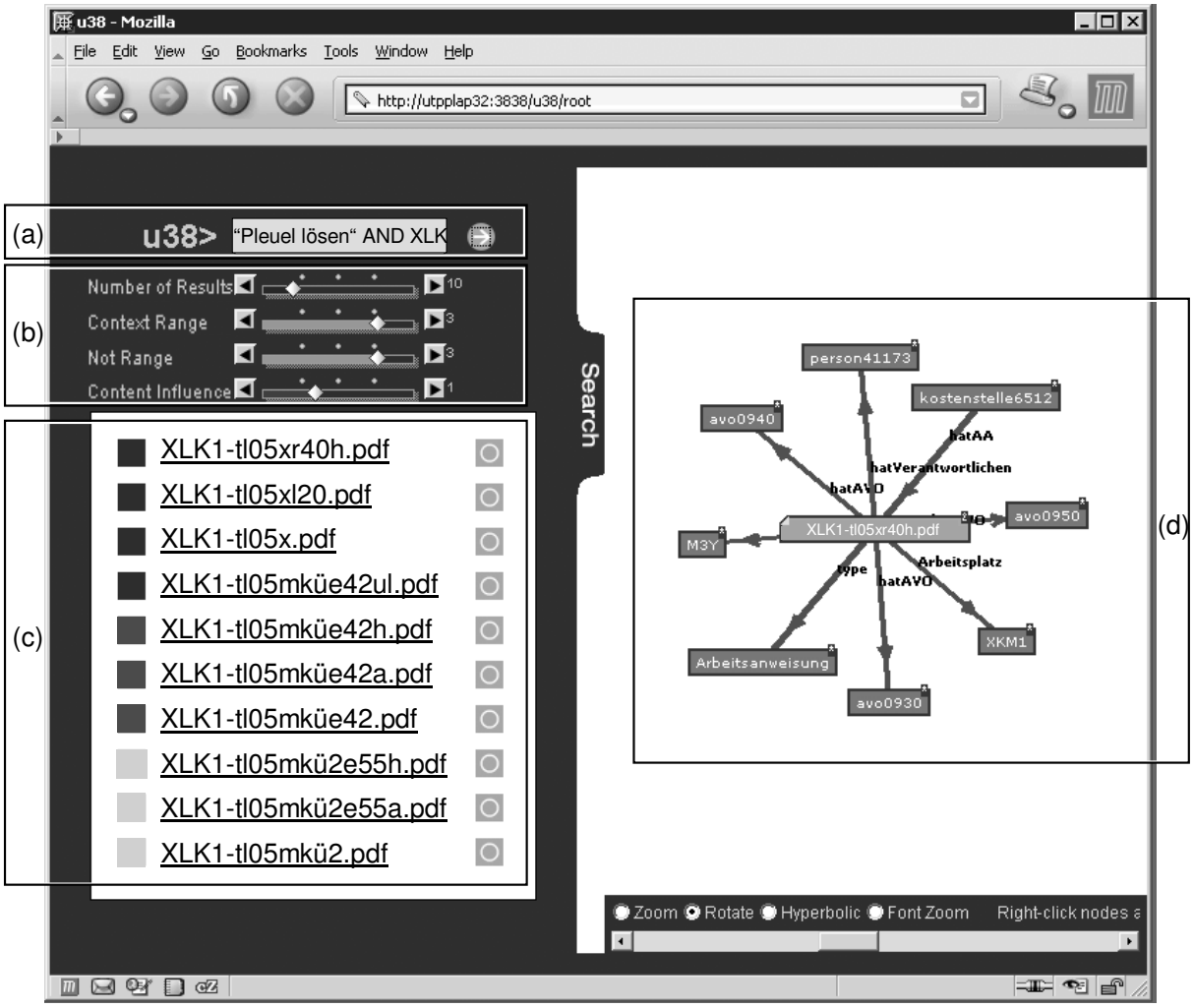


Abbildung 4.8: Benutzerschnittstelle des Prototypen. (a) Eingabefeld. (b) Parametereingabe. (c) Resultatliste. (d) Interaktive Darstellung des ContextGraphs.

Für die Beispielanfrage "Pleuel lösen" AND XLK1 ist diese Erklärung in Abbildung 4.9 für das am höchsten bewertete Dokument dargestellt. Dort ist zu sehen, dass die Phrase Pleuel lösen lediglich im Text des Dokuments vorkommt, nicht jedoch im Kontext, während das Kürzel XLK1 sowohl im Kontext als auch im Text des Dokuments enthalten ist.

4.9 Zusammenfassung

In diesem Kapitel wurde der im Rahmen der vorliegenden Arbeit implementierte Protoyp beschrieben. Zu Beginn wurde eine komponentenbasierte und somit leicht anpassbare und erweiterbare Architektur für eine indexbasierte Suchmaschine präsentiert. Das System zerfällt in zwei Teile. Zur Indexierungszeit werden vom Indexer die zu durchsuchenden Dokumente und Datenbanken analysiert und im Index abgelegt. Dieser Index ist die Grundlage des Searchers, der zur Anfragezeit die Suchanfragen der Benutzer abarbeitet. Beide Teile wurden komponentenweise vorgestellt

Score Details for Document 0 - Mozilla

Explaining Document 0

(http://as.informatik.uni-stuttgart.de/as/u38/XLK1-tl05xr40h.pdf)

Expand All Collapse All

```
1.0119991 = sum of:
    0.18653868 = product of:
        0.93269336 = weight(DOCTEXT:"pleuel lösen" in 118), product of:
            0.24516422 = queryWeight(DOCTEXT:"pleuel lösen"), product of:
                6.086979 = idf(DOCTEXT: pleuel=42 lösen=14)
                0.04027683 = queryNorm
            3.8043618 = fieldWeight(DOCTEXT:"pleuel lösen" in 118), product of:
                1.0 = tf(phraseFreq=1.0)
                6.086979 = idf(DOCTEXT: pleuel=42 lösen=14)
                0.625 = fieldNorm(field=DOCTEXT, doc=118)
        0.2 = coord(1/5)
    0.82546043 = product of:
        1.0318255 = sum of:
            0.64167845 = weight(RELATED_01:xlk1 in 118), product of:
                0.14379089 = queryWeight(RELATED_01:xlk1), product of:
                    3.5700645 = idf(docFreq=14)
                    0.04027683 = queryNorm
                4.4625807 = fieldWeight(RELATED_01:xlk1 in 118), product of:
                    1.0 = tf(termFreq(RELATED_01:xlk1)=1)
                    3.5700645 = idf(docFreq=14)
                    1.25 = fieldNorm(field=RELATED_01, doc=118)
            0.1706301 = weight(RELATED_02:xlk1 in 118), product of:
                0.11723855 = queryWeight(RELATED_02:xlk1), product of:
                    2.9108188 = idf(docFreq=28)
                    0.04027683 = queryNorm
                1.4554094 = fieldWeight(RELATED_02:xlk1 in 118), product of:
                    1.0 = tf(termFreq(RELATED_02:xlk1)=1)
                    2.9108188 = idf(docFreq=28)
                    0.5 = fieldNorm(field=RELATED_02, doc=118)
            0.006229365 = weight(RELATED_03:xlk1 in 118), product of:
                0.04007186 = queryWeight(RELATED_03:xlk1), product of:
                    0.99491096 = idf(docFreq=196)
                    0.04027683 = queryNorm
                0.15545484 = fieldWeight(RELATED_03:xlk1 in 118), product of:
                    1.0 = tf(termFreq(RELATED_03:xlk1)=1)
                    0.99491096 = idf(docFreq=196)
                    0.15625 = fieldNorm(field=RELATED_03, doc=118)
            0.21328762 = weight(DOCTEXT:xlk1 in 118), product of:
                0.11723855 = queryWeight(DOCTEXT:xlk1), product of:
                    2.9108188 = idf(docFreq=28)
                    0.04027683 = queryNorm
                1.8192618 = fieldWeight(DOCTEXT:xlk1 in 118), product of:
                    1.0 = tf(termFreq(DOCTEXT:xlk1)=1)
                    2.9108188 = idf(docFreq=28)
                    0.625 = fieldNorm(field=DOCTEXT, doc=118)
        0.8 = coord(4/5)
```

„Pleuel lösen“ im Text

XLK1 in rel_1

XLK1 in rel_2

XLK1 in rel_3

XLK1 im Text

Done

Abbildung 4.9: Bewertung eines Dokuments.

und diskutiert.

Ein besonderer Augenmerkt lag auf alternativen Implementierungen der Komponenten zur Indexerzeugung und Indexauswertung. Der Einzelschrittindex zur Schlüsselwortsuche bildet die Suchbegriffe einer Anfrage direkt auf Dokumente ab. Die im Einzelschrittindex enthaltene Redundanz wird im Doppelschrittindex zur Schlüsselwortsuche vermieden. Hier werden Suchbegriffe im ersten Schritt auf die Knoten des ContextGraphs abgebildet und diese dann im zweiten Schritt für eine Suche nach Dokumenten eingesetzt. Der Doppelschrittindex für Ähnlichkeitssuche schließlich ist als Erweiterung des Doppelschrittindex für Schlüsselwortsuche zu sehen, auf dem zusätzlich eine Suche nach ähnlichen Dokumenten durchgeführt werden kann.

In diesem Kapitel wurden einige Erweiterungen des Systems angesprochen.

- Funktionalitäten. Am Markt verfügbare, Text-basierte Suchmaschinen bieten

eine Vielzahl von Funktionalitäten – z.B. die oben angesprochene Höherbewertung von Begriffen in Überschriften. Viele dieser Funktionalitäten lassen sich im Rahmen der vorgestellten Architektur leicht realisieren. Da sie nicht im Fokus der Untersuchungen der vorliegenden Arbeit stehen, wurden sie jedoch nicht weiter beachtet.

- Crawler. Um inkrementelle Index-Anpassungen bei neu hinzukommenden oder sich ändernden Daten durchführen zu können, ist die Entwicklung einer Crawler-Komponente nötig.

- Import-Komponenten. Neben dem Import von relationalen Datenbanken und Ontologien könnten weitere Importkomponenten z.B. für XML-Daten entwickelt werden.

- Semantische Suche. Das System kann um verschiedenste Aspekt der semantischen Suche erweitert werden. Dazu zählt nicht nur die ontologiebasierte Modifikation der Anfrage, sondern auch z.B. die Berücksichtigung des Benutzerkontexts.

Alle diese Erweiterungen sind denkbar, wenn nicht sogar wünschenswert. Der in diesem Kapitel vorgestellte Prototyp wurde mit dem Ziel implementiert, die Auswirkungen der Übertragung der Konzepte textbasierter Suche auf das kontextbasierte Szenario zu untersuchen. Die Auswertung dieser Untersuchung wird im nächsten Kapitel vorgestellt.

5

Analyse und Bewertung

In diesem Kapitel werden die Auswirkungen der Erweiterung textbasierter Suche auf das kontextbasierte Szenario anhand der in Kapitel 3 eingeführten Konzepte und des in Kapitel 4 vorgestellten Systems analysiert und bewertet. Dazu werden zuerst die Szenarien und die Messumgebung und sodann die einzelnen Messungen behandelt. Im Einzelnen werden Messreihen zur Qualität der Suchresultate, zur Größe des erstellten Index, zur Laufzeit für die Berechnung der kürzesten Wege im ContextGraph, zur Laufzeit für die Indexerstellung und zur Laufzeit von Suchanfragen präsentiert.

Wie bei Suchmaschinen üblich, existieren auch im hier analysierten System mehrere Parameter, die die Funktionsweise der Suchmaschine beeinflussen, so z.B. die Kantengewichtung im ContextGraph, die Kontextgröße $cRange$ und der regulierbare Einfluss von Text und Kontext auf die Resultatsbewertung nach Definition 3.2.4. Es liegt auf der Hand, dass im Rahmen der vorliegenden Arbeit nicht alle Belegungen aller Parameter umfassend evaluiert werden können. Das folgende Kapitel fokussiert auf die Belegung des für die vorliegende Arbeit zentralen Parameters zur Bestimmung der betrachteten Kontextgröße $cRange$. Die Einstellung dieses Parameters hat nicht nur Einfluss auf die Qualität der Suchresultate, sondern auch auf die Speicheranforderungen und die Laufzeit des Systems.

5.1 Szenarien und Messumgebung

Im Rahmen der vorliegenden Arbeit wurde der Prototyp anhand zweier Szenarien erprobt, die im Folgenden kurz vorgestellt werden.

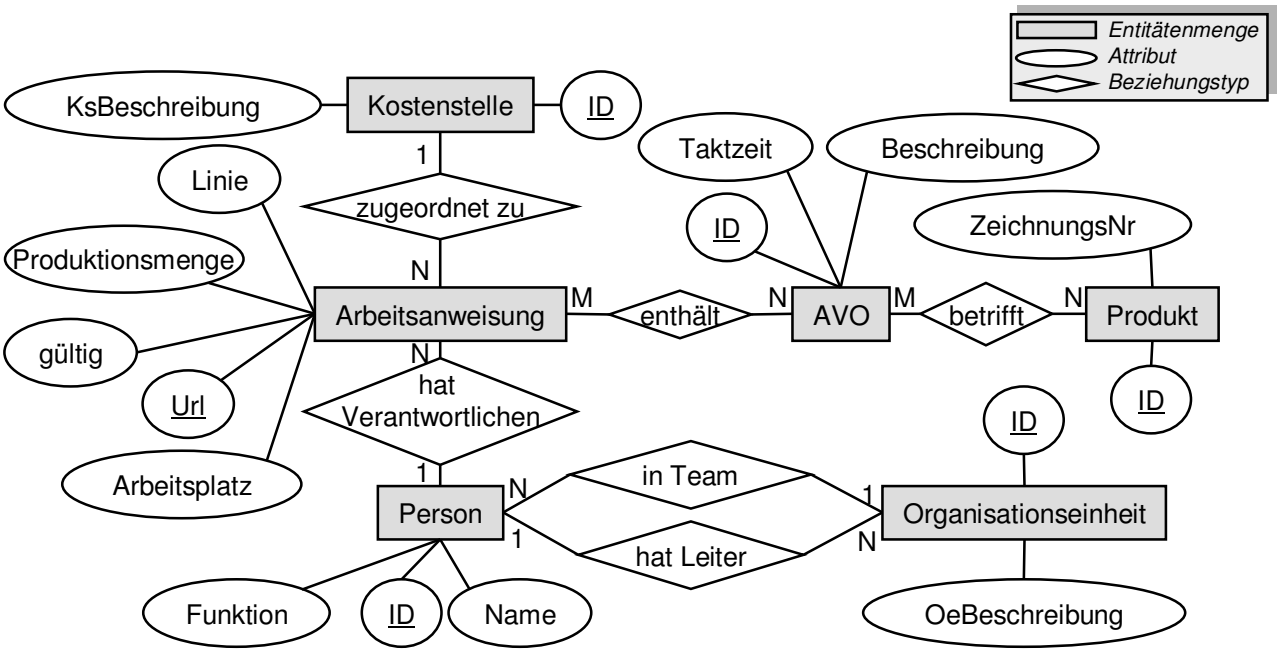


Abbildung 5.1: ER-Diagramm des Maschinenbauszenarios.

5.1.1 Maschinenbauszenario

Das erste Szenario besteht aus (verfremdeten) echten Dokumenten und Datenbankdaten eines Autobauers und entspricht somit den Randbedingungen der vorliegenden Arbeit. Es wurde v.a. dafür verwendet, die Qualität der Suchresultate zu bewerten.

Das Szenario enthält eine Tabelle eines Dokumentenmanagementsystems mit 196 Tupeln bzw. Verweisen auf Dokumente, sowie sieben weiteren Tabellen. Zusammengenommen beinhalten die Tabellen 25 Attribute und 2107 Tupel. Das entsprechende ER-Diagramm ist in Abbildung 5.1 dargestellt. Die Entitätenmenge Arbeitsanweisung repräsentiert die im System hinterlegten Dokumente. Im vorliegenden Szenario beschreibt eine Arbeitsanweisung diejenigen Arbeitsschritte, die an einem bestimmten Montageplatz nacheinander durchgeführt werden müssen. Jede Arbeitsanweisung ist einer Kostenstelle sowie einer verantwortlichen Person zugeordnet. Die in einer Arbeitsanweisung vorgegebenen Schritte werden Arbeitsvorgangsbeschreibung, kurz: AVO, genannt. Dabei kann eine AVO in mehreren Arbeitsanweisungen enthalten sein. Jede AVO kann auf mehrere Produkte angewandt werden. Die im System vermerkten Personen sind zu Organisationseinheiten zusammengefasst, die jeweils von einer Person geleitet werden.

Nach dem Import der Datenbank in einen ContextGraph wurde das folgende einfache Kantengewichtungsschema angewandt:

- Kanten, die durch Attributbeziehungen entstehen, erhalten das Gewicht 0 (null).
- Kanten, die Fremdschlüsselbeziehungen repräsentieren, werden mit 1 (eins) gewichtet.
- Kanten, die die Beziehungen zwischen Tupeln und Tabellen anzeigen, werden in Hin- bzw. Rückrichtung mit 1 (eins) bzw. 4 gewichtet.

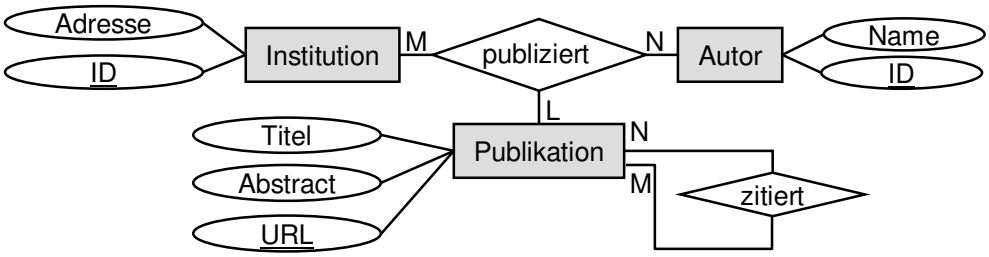


Abbildung 5.2: ER-Diagramm des Citeseerszenarios.

Die Kantengewichtung wird somit nach der Kategorisierung in Abschnitt 3.1.3.2 manuell und auf Schemaebene durchgeführt, wobei auf die Möglichkeit zur Zuweisung attributsspezifischer Kantengewichte verzichtet wird. Die Belegung der Attributskanten mit 0 (null) und der Fremdschlüsselkanten mit 1 (eins) bildet den in Abschnitt 3.1.2 diskutierten Ansatz nach, bei dem die gesamte Information eines Tupels in einem einzigen Knoten abgespeichert wird.

5.1.2 Citeseerszenario

Da das im letzten Abschnitt beschriebene Maschinenbauszenario zwar Aufschlüsse über die Qualität der Suchergebnisse geben kann, jedoch für Messungen der Laufzeit und des Ressourcenverbrauchs zu klein ist, wurde ein weiteres Szenario aus dem Dokumentenstamm der Suchmaschine Citeseer[1] generiert.

Im Citeseer-Archiv sind wissenschaftliche Veröffentlichungen verzeichnet, aus denen Metadaten zur Überschrift, Kurzzusammenfassung, zu Autoren und zu den Institutionen der Autoren extrahiert wurden. Um das Szenario für die vorliegende Arbeit zu erstellen, wurden diese Metadaten aufbereitet und in eine einfache Datenbank geladen. Das ER-Diagramm der Datenbank ist in Abbildung 5.2 dargestellt. Eine Publikation wird demnach von mehreren Autoren verfasst, die verschiedenen Institutionen zugeordnet sein können. Außerdem zitieren sich Publikationen untereinander. Die im Citeseer-Archiv ebenfalls verfügbare Information, auf welchen Servern zu welcher Zeit die Publikationen abrufbar waren, wurden für das hier vorgestellte Szenario nicht beachtet.

Da in dem so gewonnenen Datenbestand viele Dokumente mehrfach repräsentiert sind, wurden Duplikate so weit wie möglich entfernt. Anschließend wurden zu den Metadaten die entsprechenden Dokumente (soweit vorhanden) aus dem Web heruntergeladen und ins TXT-Format konvertiert. Auf diese Weise entstand ein Datensatz mit 74.854 Dokumenten. Um die Skalierbarkeit des Systems beurteilen zu können, wurden Teilmengen verschiedener Größe aus dem Dokumentenstamm extrahiert. Damit die Metadatenstruktur der Teilmengen möglichst der Struktur des gesamten Dokumentenstamms entspricht, wurde hierfür eine Breitensuche über die Zitierungsbeziehung der Dokumente implementiert.

[1] `http://citeseer.ist.psu.edu/` [08.08.2006]

Zum Import der Datenbank in den ContextGraph wurde eine einfache, domänenspezifische Importkomponente implementiert. Beim Import der Datensätze werden hier keine Kanten eingefügt, die die Zugehörigkeit von Tupeln und Tabellen repräsentieren, da eine Suche, die die Begriffe "Institution", "Autor" oder "Publikation" enthält, von dieser Information wohl kaum profitieren dürfte und zwei Entitäten, egal welcher Menge, nur auf Basis der Zugehörigkeit zur gleichen Menge kein semantischer Zusammenhang zuzuschreiben ist. Weiterhin werden keine Kanten eingefügt, die von Knoten, die Tupelattribute repräsentieren, zu ihren jeweiligen Tupelknoten zeigen, da diese Art von Kanten zur Kontextbestimmung irrelevant sind. Der domänenspezifische DB-Import ermöglicht die Anwendung eines konstanten Kantengewichtungsschemas, bei dem alle Kanten gleichermaßen mit 1 (eins) bewertet werden. Die Merkmale der so entstandenen Datensätze sind in Tabelle 5.1 aufgeführt. Für jeden Datensatz sind die Anzahl der Dokumente und die Anzahl der Kanten und Knoten des jeweiligen ContextGraphs angegeben.

Datensatz	Anzahl der Dokumente	Anzahl der Kanten	Anzahl der Knoten
10k	10.000	117.070	77.992
20k	20.000	223.461	143.954
30k	30.000	322.748	203.881
40k	40.000	419.644	260.689
50k	50.000	512.667	315.798
60k	60.000	605.308	370.787
70k	70.000	702.391	430.926

Tabelle 5.1: Charakteristika der Datensätze des Citeseer-Szenarios.

Einen Überblick über das Anwachsen der Menge der gefundenen Pfade bei der Bestimmung der kürzesten Wege gibt Abbildung 5.3. Im linken Diagramm (a) ist zu sehen, dass die Anzahl der Pfade für eine feste Belegung von $cRange$ mit zunehmender Datensatzgröße leicht sublinear zunimmt. Im rechten Diagramm (b) ist die Zunahme der Pfade für feste Datensätze mit zunehmendem $cRange$ abgebildet. Beide Diagramme werden für die spätere Diskussion in Abschnitt 5.3 noch von Interesse sein.

5.1.3 Messumgebung

Das für die Messungen eingesetzte System bestand aus einem AMD Quad Opteron Server mit 32GB RAM mit Centos Linux Betriebssystem. Es wurde eine Java Virtual Machine in der 64-bit Ausführung der Version $1.5.0_06$ eingesetzt. Beim Aufruf wurde die Heap-Größe der Virtual-Machine auf 24GB begrenzt. Als Datenbank kam IBMs DB2 in der 64-bit Variante der Version 08.02.0004 zum Einsatz. Trotz der

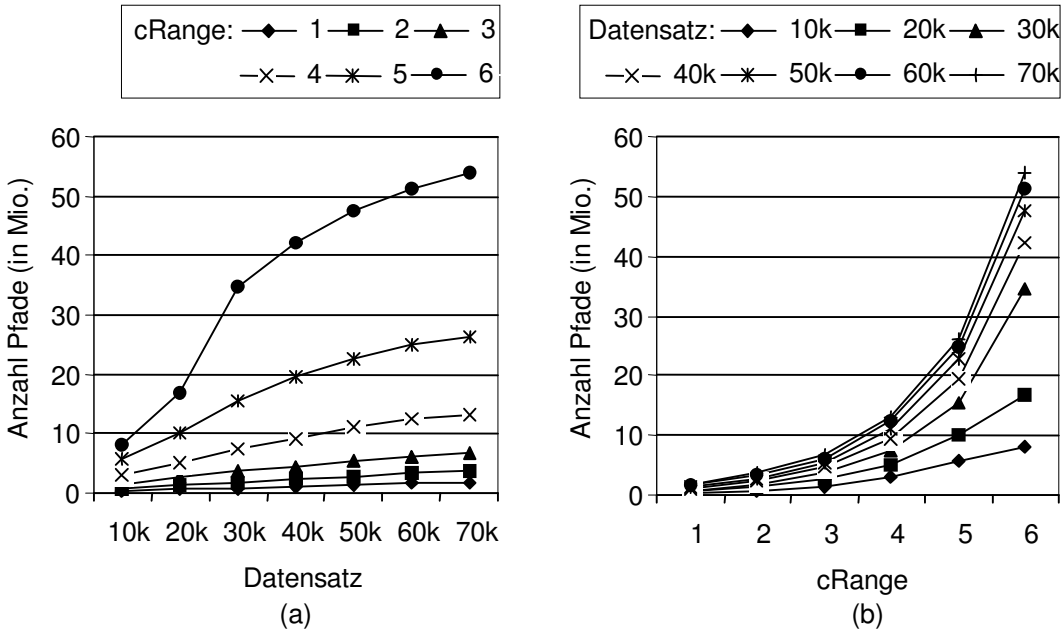


Abbildung 5.3: Anzahl der kürzesten Wege. (a) Für festes $cRange$, variable Datensatzgröße. (b) Für feste Datensätze, variables $cRange$.

NUMA-Architektur des verwendeten Server-Systems wurde im Prototyp keinerlei Parallelisierungsstrategie angewendet, sofern sie nicht in externen Programmbibliotheken schon vorhanden war.

5.2 Qualität der Suchresultate

In diesem Abschnitt wird die Qualität der Suchresultate diskutiert. Dazu wird zunächst das zur Messung der Qualität eingesetzte Maß Precision@10 erläutert. Darauf aufbauend wird anschließend die Qualität der Schlüsselwortsuche sowie die Qualität der Suche nach ähnlichen Dokumenten analysiert.

5.2.1 Das Qualitätsmaß Precision@10

Zur Messung der Qualität der Suchergebnisse kommt das Qualitätsmaß Precision@10 (auch: P_{10}, P@10) zum Einsatz [Witten et al., 1999, S.189]. Precision@10 ist ein häufig verwendetes Qualitätsmaß, siehe z.B. [Cohen et al., 2003; Haveliwala, 2002]. Im Gegensatz zu der in Abschnitt 2.1.4 definierten Precision wird es definiert als der Anteil der ersten x Suchresultate, welche für die Anfrage relevant sind:

$$\text{Precision@}x = \frac{\text{Anzahl der relevanten Dokumente unter den ersten } x \text{ Treffern}}{\min\{x, \text{Anzahl der gefundenen Dokumente}\}}$$

Man spricht an dieser Stelle oftmals vom “Cutoff-Level” (etwa: Absperrebene) x, da nur die ersten x Resultate einer Anfrage für die Qualitätsbewertung beachtet werden.

Das Maß Precision@10 ist zur Bewertung der Qualität der Suchresultate im vorliegenden Fall aus den folgenden beiden Gründen angemessen.

Zum einen spiegelt Precision@10 das typische Verhalten des Benutzers einer Suchmaschine wider. Laut Studien [Jansen et al., 1998; Wolfram, 2000] beachtet der durchschnittliche Benutzer einer Suchmaschine häufig nur die erste Seite der Ergebnisse seiner Suchanfrage. Enthalten diese Ergebnisse kein für ihn relevantes Dokument, so wird eine neue, geänderte Anfrage gestellt. In diesem Szenario ist daher von Interesse, wieviele relevante Suchergebnisse unter den ersten x Treffern zu finden sind. Finden sich alle relevanten Dokumente am Schluss der Ergebnisliste, so werden sie vom Anwender erst gar nicht beachtet. Die Zufriedenheit des Nutzers und somit die Qualität der Suchmaschine wird daher an der Anzahl der relevanten Dokumente unter den ersten Treffern gemessen [Witten et al., 1999, S. 197].

Der zweite Grund, warum Precision@10 in diesem Szenario ein ausreichendes Qualitätsmaß darstellt, liegt in der Arbeitsweise der Suchmaschine. Wie bereits in Abschnitt 2.1.4 angesprochen, stellen Precision und Recall konkurrierende Maße dar. Jede Maßnahme zur Verbesserung der Precision einer Suchmaschine läuft stets das Risiko eines abnehmenden Recalls und umgekehrt. Im vorliegenden Szenario werden zu jedem bestehenden Dokument zusätzliche Suchbegriffe im Index hinterlegt; andersherum betrachtet kann jedes Dokument somit über eine größere Anzahl von Suchbegriffen gefunden werden. Daher kann die Suchmaschine für jeden Suchbegriff eine größere Anzahl von Dokumenten lokalisieren. Für Anfragen ohne negierte Suchbegriffe hat somit die kontextbasierte Suche stets mindestens den gleichen Recall wie textbasierte Suche. Seien A_k bzw. A_t die Resultat- oder Antwortmengen der kontextbasierten bzw. textbasierten Suchmaschine. Dann gilt $A_t \subseteq A_k$. Seien weiterhin R die Menge der relevanten Dokumente sowie Recall_k, bzw. Recall_t der Recall der kontext- bzw. textbasierten Suche. Dann gilt:

$$\begin{aligned}
\text{Recall}_k &= \frac{|A_k \cap R|}{|R|} \quad \text{, nach der Def. von Recall in Abschnitt 2.1.4} \\
&= \frac{|(A_k - A_t) \cap R| + |A_t \cap R|}{|R|} \\
&\geq \frac{|A_t \cap R|}{|R|} = \text{Recall}_t
\end{aligned}$$

Dabei steht $(A_k - A_t)$ für die Menge von Suchresultaten, die zwar in A_k, jedoch nicht in A_t enthalten sind.

Für Anfragen mit negierten Suchbegriffen gilt diese Überlegung nicht a priori. Der Algorithmus zur adaptiven Kontextgröße aus Abschnitt 3.2.1.5 lässt sich jedoch leicht dahingehend modifizieren, dass auch hier der Recall nicht abnimmt. Dazu wird lediglich die Schleife bei nicht-leerer Resultatsmenge nicht abgebrochen. Die Liste der Resultate wird dann aus einer Konkatenation der einzelnen Suchen mit abnehmendem $nRange$ gebildet.

Somit nimmt durch die Beachtung von Dokumentenkontext der Recall einer Anfrage niemals ab. Gegenüber allen aus Precision und Recall zusammengesetzten Maßen ist die alleinige Untersuchung der Precision also als eine untere Schranke der Qualität zu sehen. Obwohl evtl. mit kombinierten Maßen größere Qualitätsvorteile hätten demonstriert werden können, wurde dem intuitiv leicht nachvollziehbaren, reinen Precision-Maß der Vorrang eingeräumt.

5.2.2 Schlüsselwortsuche

Zur Qualitätsbewertung der Schlüsselwortsuche wurden Experimente in beiden Szenarien durchgeführt. In diesem Abschnitt wird zunächst die Art der Experimente beschrieben, sodann wird auf die Messreihen zu jedem Szenario eingegangen.

5.2.2.1 Experiment

Alle Anfragen wurden mit unterschiedlichen Belegungen des $cRange$-Parameters durchgeführt. Die Belegung variierte dabei im Bereich zwischen 0 und 6. Der so zustandegekommenen Messreihe mit $cRange$=0 kommt eine besondere Bedeutung zu, da sie das Verhalten einer herkömmlichen Textsuchmaschine wiedergibt, welche keinerlei Kontextinformation in die Verarbeitung der Anfragen miteinbezieht. Um die Qualität der Messreihe mit $cRange$=0 zu relativieren, wurden zusätzlich Experimente mit der frei erhältlichen Desktopsuchmaschine Google Desktop[2] durchgeführt. Auch sie verwendet keinerlei Kontextinformation zur Auswertung von Suchanfragen.

5.2.2.2 Maschinenbauszenario

Im Maschinenbauszenario wurden 40 Anfragen an die Suchmaschine abgesetzt. Jede Anfrage enthält ein bis neun Suchbegriffe, die durch die booleschen Operatoren UND, ODER und NICHT verknüpft sind. Um Precision@10 bestimmen zu können, wurden zu jeder Anfrage die zehn Resultatdokumente mit dem höchstem Ranking manuell auf ihre Relevanz untersucht.

Um eine aussagekräftige Auswertung der Messergebnisse zu ermöglichen, wurde zwischen zwei Klassen von Suchanfragen unterschieden.

- Kontextanfragen. Eine Suchanfrage gehört zur Klasse der Kontextanfragen, falls mindestens ein Suchbegriff in keinem Dokument des Dokumentenstamms vorkommt, dafür aber im Datenbankkontext. In der Notation der Definition von $\mathrm{ctf.icf}$ (Abschnitt 3.2.1.3) gilt für eine Kontextanfrage $q = \{t_1, \ldots, t_n\}$ aus n Anfragebegriffen: $\exists\, 1 \leq i \leq n : t_i \in \mathbb{T}^{\mathbb{C}} \setminus \mathbb{T}^{\mathbb{D}}$.

[2]Bei der Bewertung des Maschinenbauszenarios kam Google Desktop in Version 20051208-de zum Einsatz. Zum Zeitpunkt der Bewertung des Citeseerszenarios war die neuere Version 4.2006.0627.0443-en-pb verfügbar und wurde daher der alten Version vorgezogen. URL: `http://desktop.google.com`

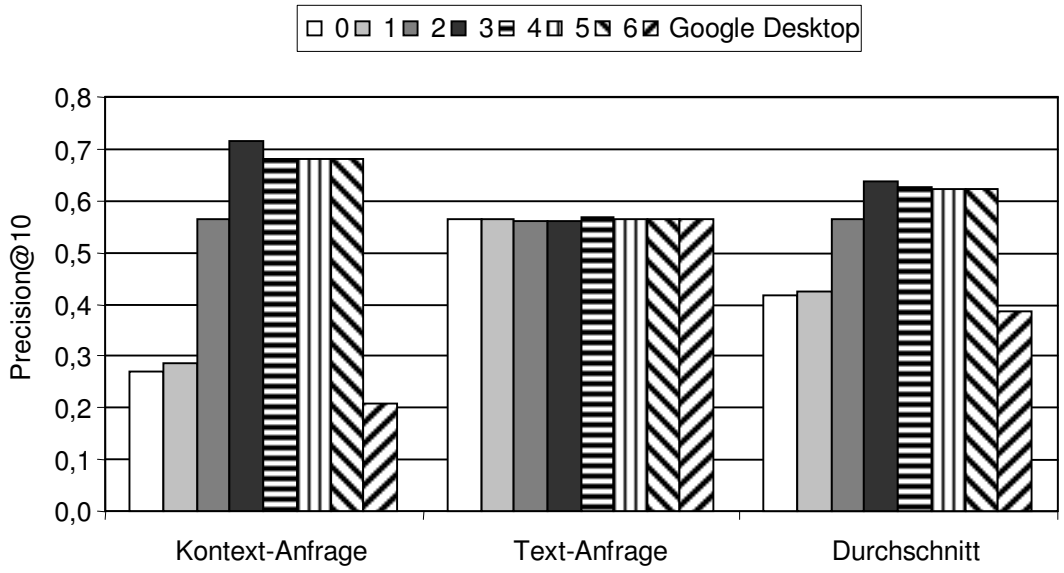


Abbildung 5.4: Qualität als Precision@10 von Suchresultaten im Maschinenbauszenario in Abhängigkeit von $cRange$ bzw. von Google Desktop für Kontext- und Textanfragen sowie für deren Durchschnitt.

- Textanfragen. Eine Suchanfrage gehört zur Klasse der Textanfragen, falls alle ihre Suchbegriffe in mindestens einem Dokument des Dokumentenstamms auftreten, bzw. $\forall\, 1 \leq i \leq n : t_i \in \mathbb{T}^{\mathbb{D}}$.

Der Sinn, Anfragen in diese beiden Klassen zu unterteilen, liegt darin, dass erwartet werden kann, dass eine kontextbasierte Suchmaschine bei Kontextanfragen gegenüber der rein textbasierten Suchmaschine einen signifikanten Vorteil aufweist. Diese Erwartung liegt darin begründet, dass für die rein textbasierte Suchmaschine ein nur im Kontext vorhandener Suchbegriff nicht existiert.

Abbildung 5.4 zeigt die Precision@10 für Kontextanfragen, für Textanfragen sowie für den Durchschnitt aller Anfragen, falls Text- und Kontextanfragen je zur Hälfte auftauchen. Die Qualität der Suchergebnisse für alle Anfragen wird für $cRange$=0 bis $cRange$=6 dargestellt und mit der Qualität von Google Desktop verglichen. Wie oben besprochen entspricht die Messreihe mit $cRange$=0 einer rein textbasierten Suchmaschine, die keinerlei Kontextinformation auswertet.

Analyse der Messergebnisse Wie erwartet zeigt die kontextbasierte Suchmaschine mit $cRange$=2 bis $cRange$=6 bei Kontextanfragen signifikante Vorteile gegenüber der textbasierten Suchmaschine ($cRange$=0). Der relativ niedrige Wert für $cRange$=1 ist darauf zurückzuführen, dass im vorliegenden Szenario in der Regel die Begriffe, welche im Kontext der Größe 1 eines Dokuments auftreten, auch schon im Dokument selbst auftreten. Enthält eine Suchanfrage einen solchen Begriff, fällt sie automatisch in die Klasse der Textanfragen. Die maximale Qualität wird erreicht für $cRange$=3; bezogen auf die reine Textsuche verbessert sich die Qualität um 165%.

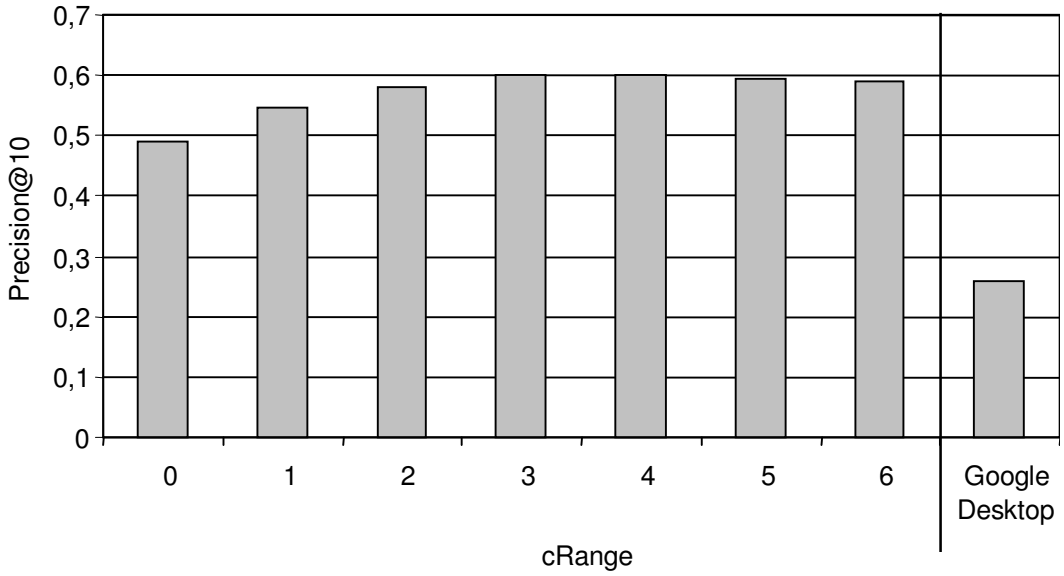


Abbildung 5.5: Qualität als Precision@10 von Suchresultaten im Citeseerszenario in Abhängigkeit von $cRange$ bzw. von Google Desktop.

Die Messwerte zur Klasse der Textanfragen zeigt, dass im vorliegenden Szenario für Anfragen die nur Begriffe enthalten, die in Dokumenten enthalten sind, kontextbasierte und textbasierte Suche gleiche Resultatqualität liefern. Dieses Ergebnis ist von Bedeutung, da es für das vorliegende Szenario impliziert, dass die kontextbasierte Suche stets mindestens die gleiche Qualität wie die textbasierte Suche aufweist. Wie die Werte für die durchschnittliche Resultatqualität anzeigen, ist in der Regel von einer deutlichen Steigerung der Qualität auszugehen; bezogen auf die reine Textsuche ergibt sich eine Verbesserung der Qualität um 53%.

Die Messungen legen wie alle weiteren in diesem Abschnitt besprochenen Messungen nahe, dass die Qualität der Ergebnisse bis zu einem Wert von $cRange$=3 oder $cRange$=4 von der Kontextinformation profitiert. Bei größeren Werten von $cRange$ kann beobachtet werden, dass die Resultatsqualität nicht weiter zunimmt. Diese Beobachtung ist wichtig, da sie die in Abschnitt 3.2.1.3 getroffene Annahme bestärkt, dass es zur Relevanzbestimmung eines Dokuments ausreicht, lediglich dessen nähere Nachbarschaft zu betrachten.

5.2.2.3 Citeseerszenario

Um das Ergebnis des Maschinenbauszenarios zu validieren, wurden im Citeseerszenario ebenfalls 40 Anfragen an das System gestellt. Da in diesem Szenario sämtliche verfügbaren Kontextdaten aus den Dokumenten extrahiert wurden, fallen alle Anfragen in die Klasse der Textanfragen. Abbildung 5.5 zeigt das Resultat der Messungen im Citeseerszenario.

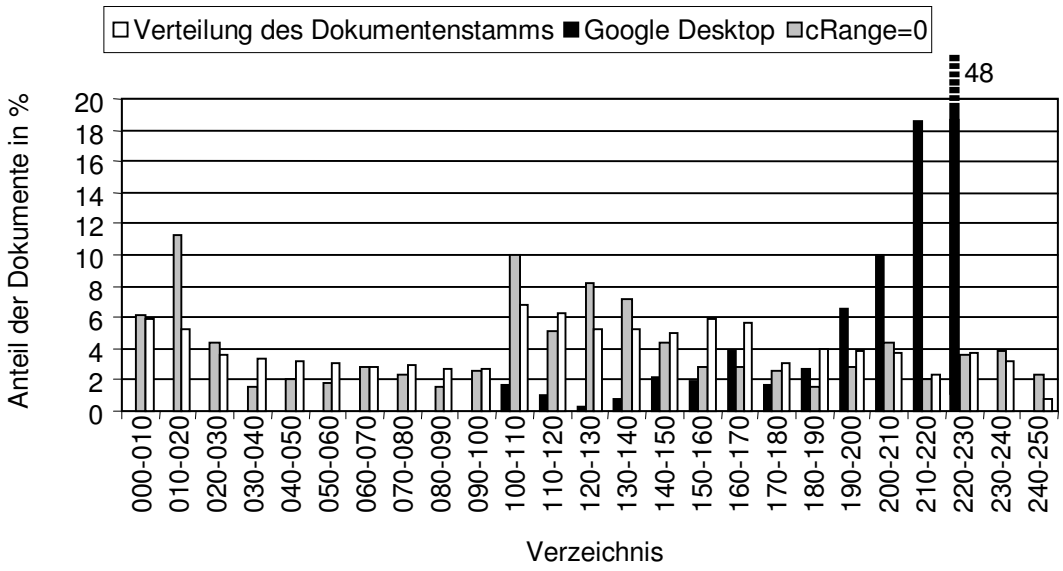


Abbildung 5.6: Verteilung der Dokumente der Resultate in Google Desktop, der Kontextsuchmaschine mit $cRange$=0 und Verteilung des Dokumentenstamms.

Analyse der Messergebnisse An den Messwerten lässt sich leicht ablesen, dass im Citeseerszenario die Miteinbeziehung des Kontexts nicht nur zu gleichbleibenden Ergebnissen führt, sondern sich die Qualität auch für Textanfragen verbessert. Diese Tatsache ist darauf zurückzuführen, dass der Kontext der Dokumente deren Titel sowie deren Kurzbeschreibung enthält und somit Suchtreffer im Kontext eines Dokuments mit hoher Wahrscheinlichkeit die Relevanz eines Dokuments anzeigen. Die maximale Qualität wird erreicht für $cRange$=3 und $cRange$=4. Im Verhältnis zur reinen Textsuche ergibt sich eine Verbesserung der Qualität um 22%.

Das vergleichsweise schlechte Abschneiden von Google Desktop überrascht an dieser Stelle, insbesondere da im Maschinenbauszenario sowie bei früheren Messungen auf kleineren Ausschnitten des Citeseerszenarios eine mit $cRange$=0 vergleichbare Qualität festgestellt wurde [Mangold et al., 2006b].

Eine Analyse der Suchergebnisse lässt vermuten, dass dieses Ergebnis darauf zurückzuführen ist, dass die Anzahl der durchsuchbaren Dokumente in Google Desktop de facto beschränkt ist. Beim Katalogisieren der Dokumente des Citeseerszenarios wurde jedem Dokument eine Identifikationsnummer zwischen 1 und 250.000 zugeteilt, die gleichzeitig als Dateiname verwendet wurde. Um die Dokumente besser handhaben zu können, wurden sie anschließend in 25 Verzeichnisse aufgeteilt, die die Namen “000-010” bis “240-250” tragen, sodass jedes Verzeichnis alle Dateien im jeweiligen Tausenderbereich beinhaltet. Abbildung 5.6 zeigt die prozentuale Verteilung des Dokumentenstamms auf die 25 Verzeichnisse.

Ebenfalls in Abbildung 5.6 ist die Verteilung der Suchergebnisse der Suchmaschine mit $cRange$=0 und von Google Desktop abgebildet. Dabei werden nur diejenigen Dokumente berücksichtigt, die auch in die Bewertung der Qualität eingehen, d.h. jeweils die ersten zehn Resultate. Bei der Analyse der von Google Desktop ge-

fundenen Dokumente fällt auf, dass aus den Verzeichnissen mit kleiner Nummer kein einziges Dokument gefunden wurde. Das Dokument mit der kleinsten Nummer liegt bereits bei über 100.000. Demgegenüber steht die Tatsache, dass die Identifikationsnummer von beinahe der Hälfte (48%) aller gefundenen Dokumente im Bereich zwischen 220.000 und 230.000 liegt. Zwar entspricht auch die Verteilung der Suchergebnisse mit $cRange$=0 nicht völlig der Verteilung des Dokumentenstamms, bei Google Desktop muss jedoch davon ausgegangen werden, dass die Ungleichverteilung der Ergebnisse in direktem Zusammenhang mit der wenig befriedigenden Resultatqualität steht. Eine weitergehende Analyse dieses Ergebnisses ist derzeit aufgrund fehlender Dokumentation der Suchmaschine nicht möglich.

5.2.3 Suche nach ähnlichen Dokumenten

Die Suche nach ähnlichen Dokumenten wurde in Abschnitt 3.2.3 eingeführt. Dort wurden $\mathrm{ctf.icf}$ und $\mathrm{nr.inf}$ als konkurrierende Maße zum Ranking von ähnlichen Dokumenten vorgestellt. Zur Bestimmung der textbasierten Ähnlichkeit wurde aus dem gegebenen Dokument ein maximal 3.000 Zeichen langes Präfix extrahiert. Abbildung 5.7 zeigt die Qualität als Precision@10 gemessen auf dem Citeseerszenario für $cRange = 1, \ldots, 6$ und die folgenden drei Varianten.

- $\mathrm{nr.inf}$ und Doppelschrittindex für Ähnlichkeitssuche. Diese Variante wurde in Abschnitt 4.6.4 beschrieben. Sie realisiert das $\mathrm{nr.inf}$-Bewertungsmaß indem zweifach auf den Dokumentenindex des Doppelschrittindex für Ähnlichkeitssuche zugegriffen wird (siehe Abbildung 4.6).

- $\mathrm{ctf.icf}$ und Einzelschrittindex. Die Begriffe des Präfixes werden in eine Anfrage an den Einzelschrittindex nach Abschnitt 4.6.2 umgewandelt. Die Ergebnisse werden somit mittels $\mathrm{ctf.icf}$ bewertet.

- Textindex. Bei der reinen Textsuche wird zu Vergleichszwecken aus den Begriffen des Dokumentenpräfix eine Suchanfrage an die reine Textsuchmaschine generiert. Ihr Ergebnis ist unabhängig von der Belegung von $cRange$.

Analyse der Messergebnisse Bei der Betrachtung der Messergebnisse fallen folgende Eigenschaften auf.

Vergleicht man die kontextbasierte Suche ($cRange \geq 1$) mit der rein textbasierten Suche, so ist festzustellen, dass der Qualitätsunterschied mit 10% Verbesserung der Suchqualität bei $\mathrm{nr.inf}$ und $cRange$=2 gegenüber der textbasierten Suche eher gering ausfällt. Dieses Ergebnis legt die Vermutung nahe, dass bei der kontextbasierten Bewertung der Suchresultate der durch die Textsuche beigesteuerte Anteil den $\mathrm{nr.inf}$-Anteil stark überwiegt (siehe Definition 3.2.6 zur kombinierten Berechnung

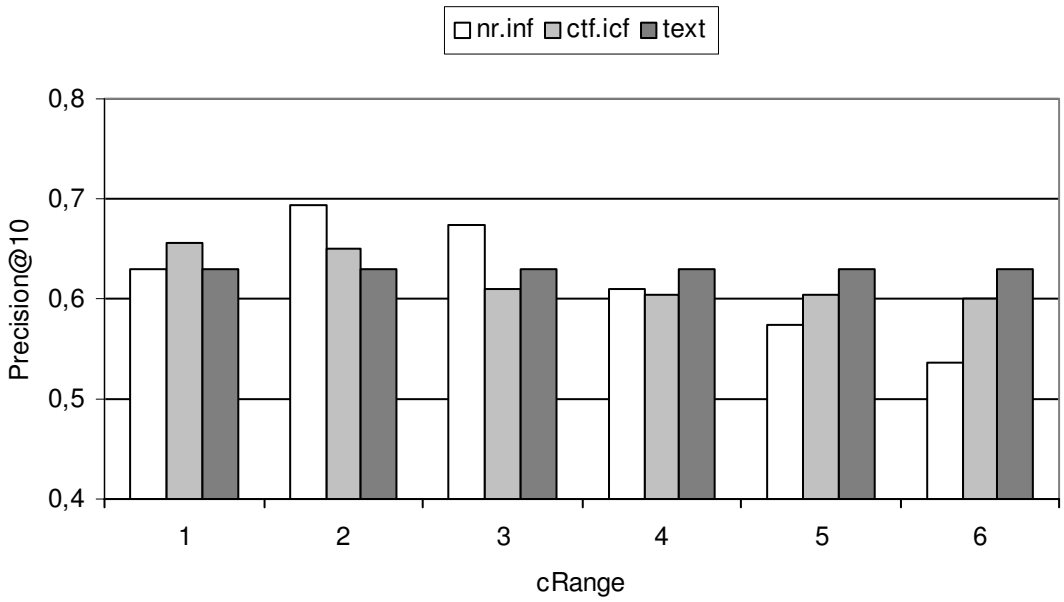


Abbildung 5.7: Qualität als Precision@10 bei der Suche nach ähnlichen Dokumenten. Vergleich von nr.inf mit ctf.icf in Abhängigkeit von $cRange$.

der Begriffsgewichte) und so weitgehend dieselben Dokumente wie bei der textbasierten Suche gefunden werden. Eine dementsprechende Untersuchung der Suchresultate widerlegt jedoch diese Vermutung. Es konnte festgestellt werden, dass beispielsweise für die Suche mit nr.inf und $cRange$=2 im Durchschnitt 5,8 der ersten zehn Dokumente ebenfalls im Ergebnis der Textsuche enthalten waren. Die Ergebnismengen beider Suchen überlappen demnach nur knapp mehr als zur Hälfte.

Weiterhin fällt auf, dass der Unterschied zwischen nr.inf und ctf.icf ebenfalls gering ausfällt. Analog zu obiger Diskussion wurde demzufolge auch für den Vergleich von ctf.icf mit nr.inf die Überlappung der Resultatmenge analysiert. Für die jeweils ersten zehn Resultate der Messreihen mit nr.inf und $cRange$=2 bzw. ctf.icf und $cRange$=4 liegt die Überlappung mit durchschnittlich 4,5 Dokumenten sogar noch unter dem Überlappungswert zwischen nr.inf mit $cRange$=2 und der Textsuche.

Zu beachten ist dabei, dass die Belegung des $cRange$-Parameters bei ctf.icf und nr.inf unterschiedliche Bedeutung hat. Bei beiden Maßen legt der $cRange$-Parameter die Größe des Kontexts der zu findenden Dokumente fest. Der Unterschied besteht darin, dass bei ctf.icf solche Knoten berücksichtigt werden, die direkt die jeweiligen Stichworte enthalten, während bei nr.inf Knoten berücksichtigt werden, die wiederum in einem Kontext der Größe $cRange$ vom Ausgangsdokument entfernt sind. Effektiv bedeutet dies, dass ein Wert von $cRange$=2 bei nr.inf einem Wert von $cRange$=4 bei ctf.icf entspricht.

Sowohl der Vergleich kontextbasierter mit textbasierter Suche, als auch der Vergleich von ctf.icf mit nr.inf zeigt, dass sich die verschiedenen Strategien bzgl. des Qualitätsmaßes Precision@10 kaum unterscheiden, d.h. bei jedem Such- bzw. Bewertungsverfahren werden in etwa gleich viele Dokumente gefunden, die einem ge-

gebenen Dokument ähnlich sind. Die Tatsache, dass für jedes Suchverfahren jeweils unterschiedliche Dokumente gefunden werden, lässt jedoch vermuten, dass die *Art* der Ähnlichkeit jeweils unterschiedlich ist. Anders formuliert liefern die verschiedenen Suchstrategien Dokumente, die vom menschlichen Benutzer aus verschiedenen Gründen als ähnlich betrachtet werden. Dieser Vermutung wurde im Rahmen der vorliegenden Arbeit jedoch nicht weiter nachgegangen.

Zusammenfassend erscheint es sinnvoll, in einer Suchmaschine kontextbasierte Ähnlichkeitssuche als Alternative zu textbasierter Ähnlichkeitssuche zu implementieren. Ein Benutzer hat dann die Wahl, die Art der Ähnlichkeit der gefundenen Dokumente zur Anfragezeit zu bestimmen. Im Szenario der Unternehmenssuche könnte dies bspw. bedeuten, dass zu einem gegebenen Fehlerbericht ähnliche Dokumente gefunden werden sollen. Wird vom Benutzer die kontextbasierte Suche gewählt, so werden Fehlerberichte derselben Maschine, desselben Autors, oder desselben Unternehmensbereichs gefunden, da die Kontexte der Fehlerberichte weitgehend überlappen. Die textbasierte Suche hingegen ist nicht an Kontextinformation gebunden und könnte bspw. Emails aus der Kommunikation mit Zulieferern finden, die dieselben Begriffe beinhalten wie der vorgegebene Fehlerbericht und somit wertvolle Hinweise enthalten. Es liegt auf der Hand, dass beide Arten von Ähnlichkeit je nach Informationsbedarf nützlich sein können oder ggf. kombiniert werden könnten.

5.3 Indexgröße

In diesem Abschnitt wird untersucht, welche Auswirkung die Berücksichtigung von Kontextinformation auf die Größe des Index hat.

Die Messungen dieses und der folgenden Abschnitte basieren jeweils auf dem Citeseerszenario. Dabei werden jeweils zwei Messungen präsentiert.

- Verschiedene Datensätze. Die erste Messung dient jeweils dazu, das Verhalten der jeweiligen Messgröße in Abhängigkeit von der Datensatzgröße darzustellen. Sie soll dazu dienen, die Skalierbarkeit des Systems zu untersuchen. Da im vorigen Abschnitt festgestellt wurde, dass die Qualität der Suchresultate ab $cRange$=4 nicht weiter zunimmt, wurde die Messung jeweils mit der festen Belegung von $cRange$=4 durchgeführt.

- Verschiedene Kontextgrößen. Die zweite Messung wurde jeweils mit dem größten Datensatz 70k für verschiedene Kontextgrößen durchgeführt.

Wird im Folgenden von “Textindex” gesprochen, so ist jeweils die kontextunabhängige Referenzimplementierung des Prototypen gemeint. Diese ist identisch mit kontextbasierter Suche ohne Kontext, d.h. $cRange$=0.

Abbildung 5.8 zeigt die Größe des Indexes für $cRange$=4 in Abhängigkeit von der Größe des Datensatzes. Dabei wird nach Abschnitt 4.6 zwischen den Varianten

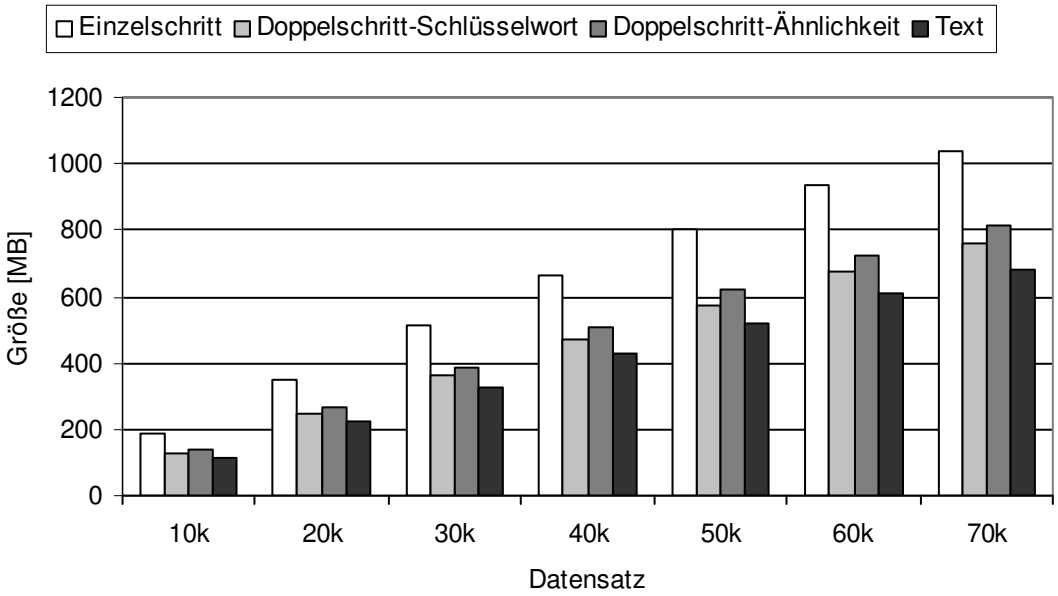


Abbildung 5.8: Indexgrößen in Abhängigkeit der Datensatzgröße für $cRange$=4. Jeweils Größe des Einzelschrittindex, des Doppelschrittindex für Schlüsselwortsuche, des Doppelschrittindex für Ähnlichkeitssuche und des reinen Textindex.

Einzelschrittindex, Doppelschrittindex für Schlüsselwortsuche und Doppelschrittindex für Ähnlichkeitssuche unterschieden. Zusätzlich zeigt der jeweils letzte Balken pro Datensatz als Referenz die Größe des reinen Textindex in dem keinerlei Kontextdaten enthalten sind.

Um die Messwerte in Abbildung 5.8 weiter zu relativieren, vergleicht Tabelle 5.2 die Größe des Einzelschrittindex mit der Größe der originalen Dokumente im PostScript-Format.

Datensatz	10k	20k	30k	40k	50k	60k	70k
Größe d. Einzelschrittindex [MB]	188	351	510	662	804	935	1.039
Größe d. Dokumentenstamms [MB]	8.657	19.007	28.006	35.909	44.423	53.518	60.622

Tabelle 5.2: Größe des Einzelschrittindex vs. Größe des Dokumentenstamms.

In Abbildung 5.9 ist die Indexgröße des Datensatzes 70k für verschiedene Belegungen von $cRange$ dargestellt. Die Größe des herkömmlichen Textindex ändert sich nicht bei variierender Belegung für $cRange$.

Analyse der Messergebnisse Wie Abbildung 5.8 zu entnehmen ist, verläuft die Größenzunahme aller Indexvarianten in etwa linear mit der Größe des Dokumentenstamms. Bei genauer Analyse der Messdaten lässt sich eine leichte Sublinearität feststellen, die wohl auf die in Zusammenhang mit Abbildung 5.3(a) diskutierte Sublinearität der Größe der Pfadmenge zurückzuführen ist.

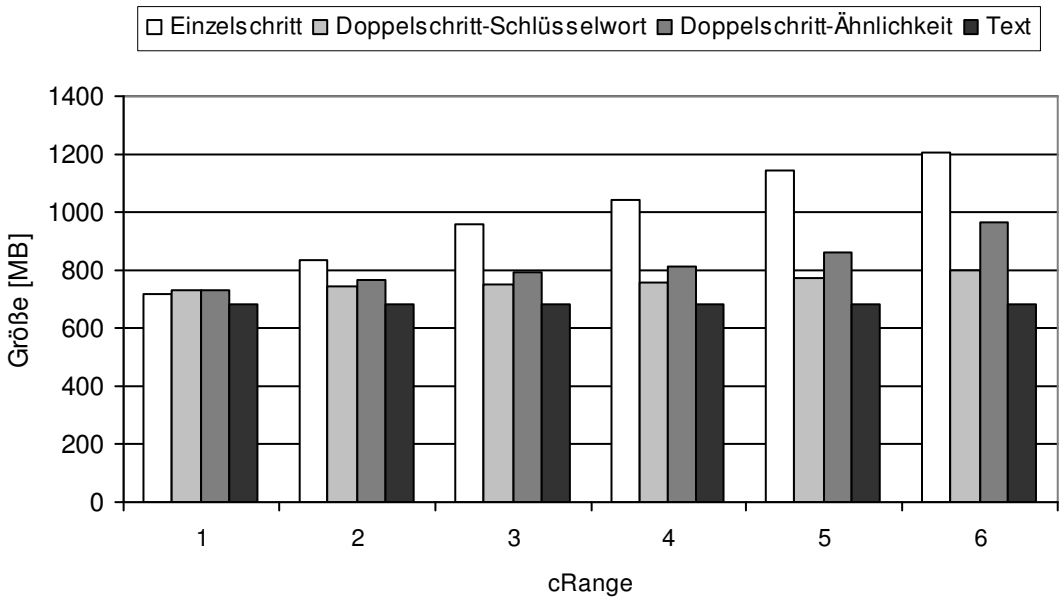


Abbildung 5.9: Indexgrößen in Abhängigkeit von $cRange$ für Datensatz 70k.

Auf den ersten Blick scheint das Verhältnis zwischen kontextbasiertem Index und reinem Textindex, d.h. der Mehraufwand der kontextbasierten Suche, ungefähr konstant zu sein. Die Analyse der Messdaten zeigt jedoch auch hier eine leichte Abnahme. So beträgt der Mehraufwand im Falle des Einzelschrittindex im Datensatz 10k 64%, im Datensatz 70k jedoch nur noch 52%. Die Ursache hierfür liegt in der Redundanz der Dokumentenkontexte. D.h. mit zunehmender Dokumentenzahl sinkt die Anzahl der pro Dokumentenkontext neu in den Index aufzunehmenden Begriffe.

Wie nach der in Abschnitt 4.6 durchgeführten Diskussion zu erwarten war, erfordert der Doppelschrittindex für Schlüsselwortsuche gegenüber dem reinen Textindex den geringsten Mehraufwand und wird vom Doppelschrittindex für Ähnlichkeitssuche und dem Einzelschrittindex gefolgt. Der Doppelschrittindex für Ähnlichkeitssuche benötigt dabei mehr Platz als der für Schlüsselwortsuche, da dort, wie in Abschnitt 4.6.4 beschrieben, der Stored-Parameter auf Ja gesetzt wird, sodass zu jedem Dokument die Knotenidentifikatoren des Kontexts aus dem Index gelesen werden können. Im Vergleich mit dem Textindex nimmt die Indexgröße der drei Implementierungen um den Faktor 1,11, 1,19 und 1,52 zu; der Mehraufwand beträgt somit 11%, 19% und 52%.

In Abbildung 5.9 ist die Größe der Indexe für verschiedene Belegungen von $cRange$ dargestellt. In etwa ist auch hier ein lineares Anwachsen der Indexgröße für zunehmendes $cRange$ festzustellen. Dies überrascht, da Abbildung 5.3(b) einen klar nicht-linearen Anstieg der Anzahl der gefundenen Pfade anzeigt. Die Ursache für dieses Phänomen liegt in der Struktur der in Lucene implementierten Indexe. Da die Überlappung der Dokumentenkontexte mit zunehmendem $cRange$ zunimmt, steigt ebenfalls die Redundanz der im Index abgelegten Information. Wie die Messergebnisse zeigen, wird diese Redundanz im Index dadurch eliminiert, dass in einem L-Field mehrfach enthaltene Begriffe lediglich durch Erhöhung eines Zählers und

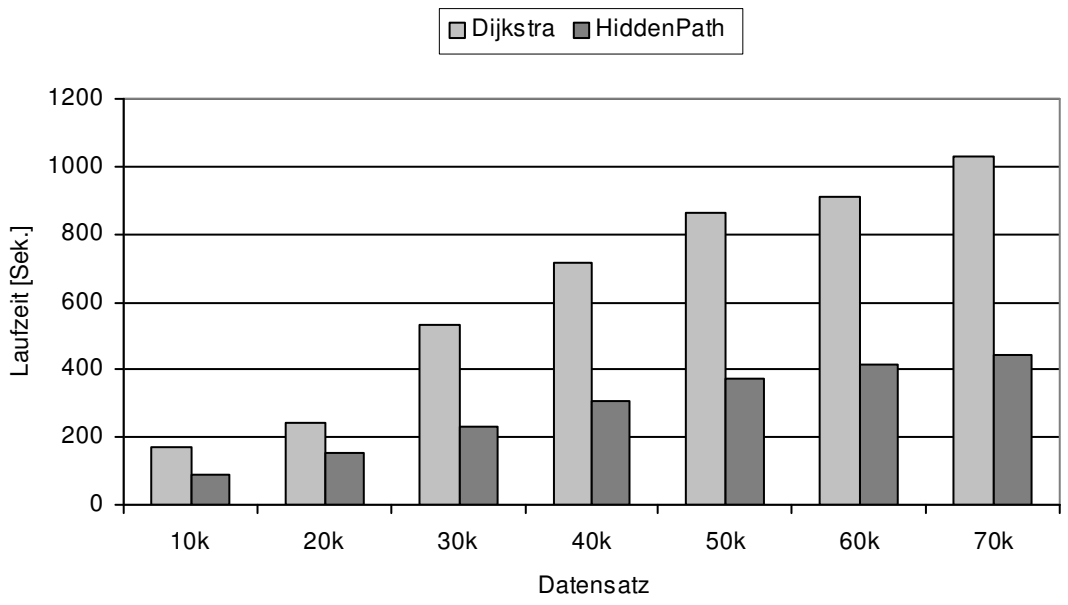


Abbildung 5.10: Laufzeit zur Berechnung der kürzesten Wege in Abhängigkeit der Datensatzgröße für $cRange$=4.

durch Abspeicherung der jeweiligen Positionen im L-Field zu Buche schlagen [Gospodnetić und Hatcher, 2005, Appendix B].

5.4 Berechnung der kürzesten Wege

Zur Berechnung der kürzesten Wege im ContextGraph wurden verschiedene Implementierungen erstellt. Abbildung 5.10 zeigt die Laufzeiten des Dijkstra-Algorithmus, bei dem nur Dokumentenknoten als Ausgangsknoten gewählt werden, sowie die Laufzeit des beschränkten HiddenPath-Algorithmus.

Bei der Implementierung des Algorithmus nach Dijkstra kommt als Implementierung der Prioritätswarteschlange die hauptspeicherbasierte Heap-Implementierung zum Einsatz, der HiddenPath-Algorithmus verwendet die externspeicherbasierte Warteschlange.

Abbildung 5.11 zeigt die Laufzeit der Algorithmen für den Datensatz 70k und verschiedene Belegungen des $cRange$-Parameters.

Analyse der Messergebnisse Die Laufzeitmessungen für veränderliche Datensatzgröße in Abbildung 5.10 zeigen einen leicht sublinearen Verlauf. Der HiddenPath-Algorithmus benötigt auf dem Datensatz 70k mit $cRange$=4 ca. 440 Sekunden, d.h. knapp über 7 Minuten. Der Dijkstra-Algorithmus benötigt für dieselbe Aufgabe 1.020 Sekunden und somit ca. 17 Minuten.

In ihrem leicht sublinearen Verlauf ähneln die gemessenen Laufzeiten stark der Entwicklung der Anzahl der gefundenen kürzesten Wege, wie in Abbildung 5.3 dargestellt. Tatsächlich liegt die Laufzeit des HiddenPath-Algorithmus quasi kon-

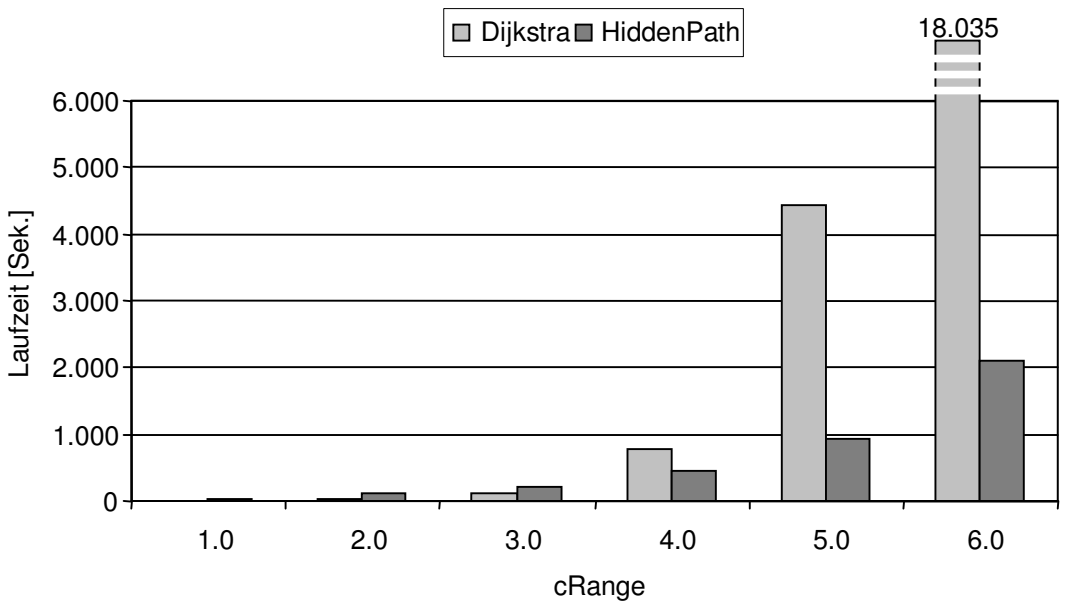


Abbildung 5.11: Laufzeit des HiddenPath-Algorithmus in Abhängigkeit von $cRange$ für Datensatz 70k.

stant bei 0,03 Millisekunden pro Pfad. Der Dijkstra-Algorithmus unterliegt höheren Schwankungen, pendelt sich jedoch zwischen 0,07 und 0,08 Millisekunden pro Pfad ein.

Die Laufzeitmessungen für veränderliche Belegung des $cRange$-Parameters auf dem Datensatz 70k in Abbildung 5.11 zeigen – analog zu den Laufzeiten der vorangegangenen Abbildung – einen Zusammenhang zwischen der Anzahl der gefundenen Pfade und der Laufzeit des Algorithmus. Für den HiddenPath-Algorithmus ist auch hier in erster Näherung von einer konstanten Zeit pro berechnetem Pfad zu sprechen, was für $cRange$=6 in einer Laufzeit von 2.090 Sekunden, d.h. knapp 35 Minuten, resultiert.

Der HiddenPath-Algorithmus ist somit deutlich performanter als der Algorithmus nach Dijkstra, obwohl durch ihn die Kontexte aller Knoten im Graph berechnet werden, während der Algorithmus nach Dijkstra lediglich die Kontexte der Dokumente berechnet.

5.5 Erstellung des Index

In diesem Abschnitt wird die Laufzeit zur Erstellung des Index analysiert. Die Abbildungen 5.12 und 5.13 zeigen die Messungen zur Erstellung des Index. Es wird jeweils die Laufzeit zur Erstellung des Einzelschrittindex (ES), des Doppelschrittindex für Schlüsselwortsuche (DSS), des Doppelschrittindex für Ähnlichkeitssuche (DSÄ) und des reinen Textindex (Text) aufgetragen. Die Messbalken sind dabei unterteilt in die Laufzeit der Importkomponente, die auch die Kantengewichtung enthält, in die Laufzeit zur Berechnung der kürzesten Wege und in die Laufzeit zur Indexie-

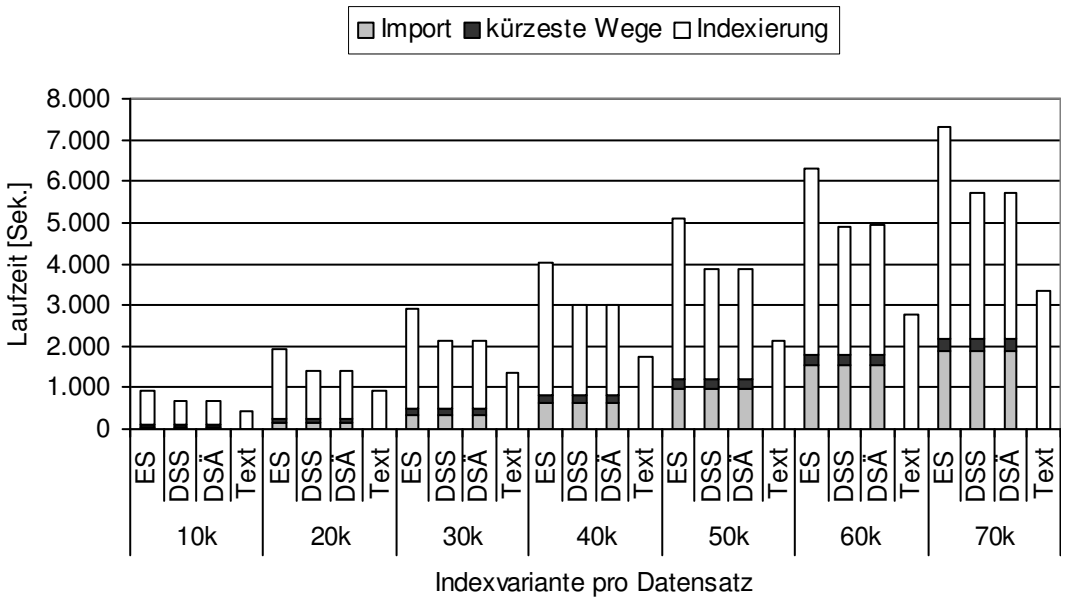


Abbildung 5.12: Laufzeit zur Indexerstellung: Einzelschrittindex (ES), Doppelschrittindex für Schlüsselwortsuche (DSS), Doppelschrittindex zur Ähnlichkeitssuche (DSÄ) und reiner Textindex (Text) in Abhängigkeit der Datensatzgröße für $cRange$=4.

rung, welche Kontextualisierung und Indexerzeugung beinhaltet. Es liegt dabei auf der Hand, dass die Laufzeiten für Import und kürzeste Wege für alle kontextbasierten Indexe identisch sind. Während Abbildung 5.12 die Laufzeit zur Indexerstellung für $cRange$=4 und die Datensätze 10k bis 70k darstellt, zeigt Abbildung 5.13 die Werte für den Datensatz 70k und $cRange$=1 bis $cRange$=6.

Um die Laufzeiten der jeweiligen Indexierungskomponenten zu verdeutlichen zeigt Abbildung 5.14 den kontextinduzierten Mehraufwand in Abhängigkeit der Datensatzgröße und für $cRange$=4 und in Abhängigkeit von $cRange$ für den Datensatz 70k.

Analyse der Messergebnisse Die Abbildungen 5.12 und5.13 zeigen in etwa einen linearen Anstieg der Indexierungszeit sowohl für größer werdende Datensätze als auch für wachsende Werte von $cRange$. Für $cRange$=4 benötigt die Erstellung des textbasierten Index für den Datensatz 70k 56 Minuten. Die Erstellung des Einzelschrittindex, des Doppelschrittindex für Schlüsselwortsuche und des Doppelschrittindex für Ähnlichkeitssuche benötigen 122, 95 und 96 Minuten, wovon jeweils 32 Minuten auf den Import der Datenbank und 4 Minuten auf die Berechnung der kürzesten Wege entfallen.

Der kontextinduzierte Mehraufwand beläuft sich demnach auf 119% für den Einzelschrittindex und 71% bzw. 72% für die Doppelschrittindexe. Der Mehraufwand zum Import der Datenbank und für die Berechnung der kürzesten Wege liegt dabei für alle Indexe bei 57% und 8%.

Hier fällt auf, dass nicht etwa die Berechnung der kürzesten Wege sondern der

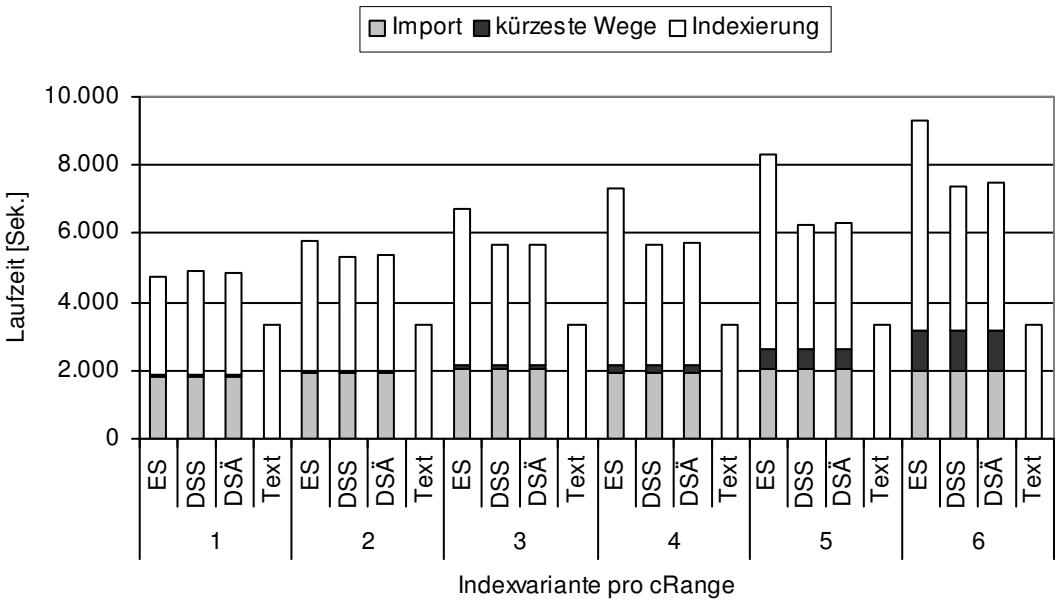


Abbildung 5.13: Laufzeit zur Indexerstellung: Einzelschrittindex (ES), Doppelschrittindex für Schlüsselwortsuche (DSS), Doppelschrittindex zur Ähnlichkeitssuche (DSÄ) und reiner Textindex (Text) in Abhängigkeit von $cRange$ für Datensatz 70k.

Import der Datenbank für großen Overhead gegenüber dem reinen Textindex verantwortlich ist. Zusätzlich entsteht Overhead bei der eigentlichen Indexierung der Daten in der Komponente zur Indexerzeugung, da im kontextbasierten Fall eine größere Anzahl von L-Fields pro Dokument abgelegt werden muss.

Abbildung 5.14 gibt einen Einblick in das Laufzeitverhältnis bei der eigentlichen Indexierung. Gemessen wurde jeweils der Mehraufwand bei der Laufzeit zur Indexierung der drei kontextbasierten Indexe bezogen auf die Laufzeit zur Indexierung des reinen Textindex.

Abbildung 5.14 (a) zeigt, dass für zunehmende Datensatzgröße der Mehraufwand für alle drei Indexarten abnimmt, was sich durch den sublinearen Verlauf der gefundenen Pfade nach Abbildung 5.3 (a) erklären lässt. Für den Doppelschrittindex für Schlüsselwortsuche liegt der Mehraufwand nur noch bei 5,9%. Der in erster Näherung lineare Verlauf des Overheads für zunehmende Werte von $cRange$ nach Abbildung 5.14 (b) steigt hingegen weniger stark an, als die Zunahme der gefundenen Pfade nach Abbildung 5.3 (b) es vermuten lassen würde. Der Verlauf erinnert eher an die Größenzunahme des Index für zunehmende Werte von $cRange$, wie in Abbildung 5.9 dargestellt. Genauso wie dort kann auch hier der flachere Verlauf auf die Art zurückgeführt werden, wie Lucene die Abspeicherung redundanter L-Fields vornimmt.

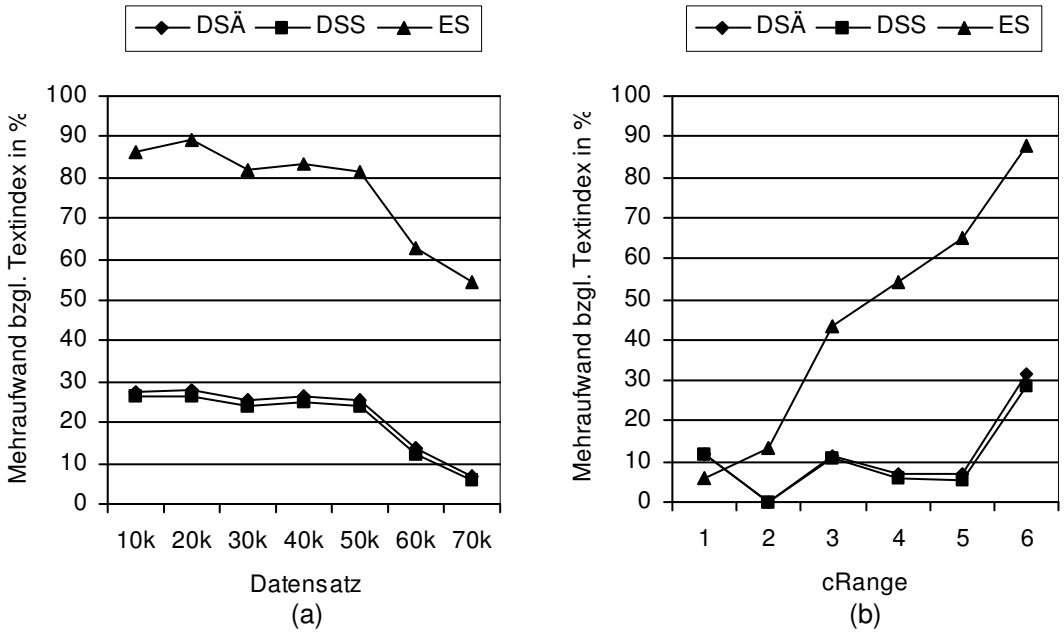


Abbildung 5.14: Kontextinduzierter Mehraufwand der Laufzeit des Indexierungsschritts. (a) In Abhängigkeit der Datensatzgröße für $cRange$=4. (b) In Abhängigkeit von $cRange$ für Datensatz 70k.

5.6 Suchanfragen

In diesem Abschnitt wird die Performanz von Suchanfragen analysiert. Gemessen wurden die Zeiten mit denselben Anfragen, die auch zur Bewertung der Qualität der Suchresultate verwendet wurden.

Die Laufzeitmessungen wurden direkt an der Komponente zur Indexauswertung vorgenommen, sodass die Overheads durch Servlet-Engine, Webserver und Netzwerk entfallen. Von der Möglichkeit, den $cRange$-Parameter zur Anfragezeit weiter einzuschränken, als zur Indexierungszeit vorgesehen, wurde kein Gebrauch gemacht. Für alle Messungen gilt der angegebene Wert von $cRange$ sowohl für die Indexierung als auch für die Anfrage. Bei der Analyse der Laufzeit von Suchanfragen wird unterschieden zwischen Schlüsselwortsuche und der Suche nach ähnlichen Dokumenten.

5.6.1 Schlüsselwortsuche

Die Laufzeiten von Schlüsselwortanfragen sind in Abbildung 5.15 und 5.16 dargestellt. Es sind jeweils die Laufzeiten an den Einzelschrittindex, an den Doppelschrittindex für Schlüsselwortsuche und als Referenz an den reinen Textindex abgebildet. Wieder zeigt Abbildung 5.15 die durchschnittliche Laufzeit pro Anfrage für wachsende Datensatzgrößen und $cRange$=4. Abbildung 5.16 zeigt die durschnittliche Laufzeit für den Datensatz 70k und wachsende Belegungen von $cRange$.

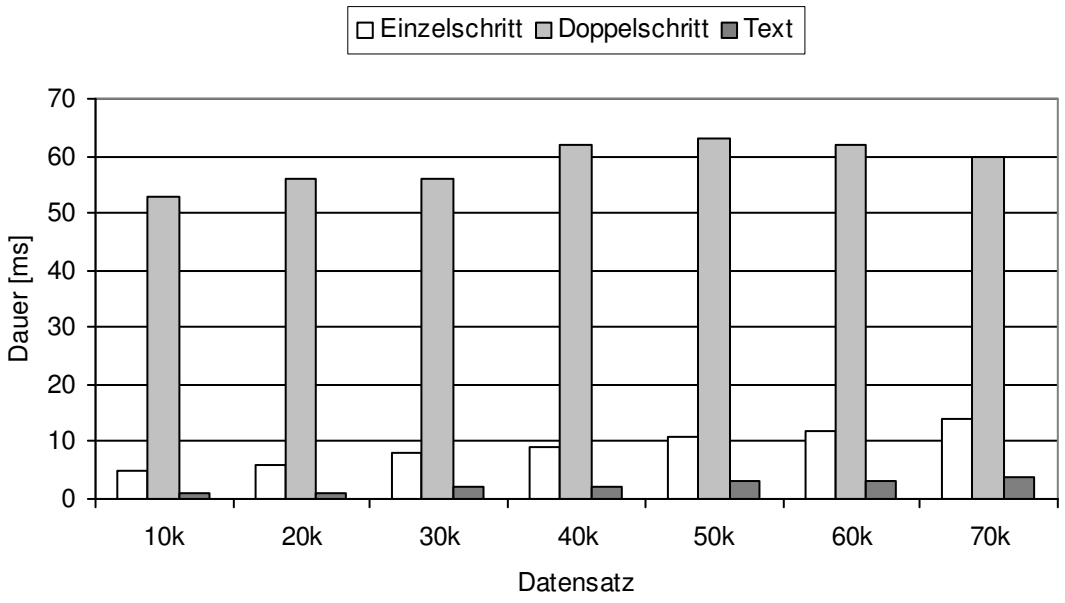


Abbildung 5.15: Dauer der Anfrageverarbeitung bei Schlüsselwortsuche. Laufzeit in Abhängigkeit der Datensatzgröße für $cRange$=4.

Analyse der Messergebnisse Die Messwerte in Abbildung 5.15 zeigen sowohl für reine Textsuche als auch für die kontextbasierte Suche im Einzelschrittindex einen sehr langsamen linearen Anstieg. Der durch die Betrachtung des Kontexts eingeführte Mehraufwand liegt hier bei ca. 200%. Die Suche im Doppelschrittindex benötigt wesentlich mehr Zeit. Ihr Mehraufwand liegt bei ca. 1400%. Die absolute Laufzeit der Anfragen aller drei Varianten liegt in jedem Fall deutlich unter 100 ms und erfüllt daher Anforderung (5) in Abschnitt 1.2 nach einer interaktiven Verwendbarkeit des Systems.

Der Overhead der beiden kontextbasierten Lösungen gegenüber der herkömmlichen reinen Textvariante ist darauf zurückzuführen, dass bei der Umformulierung der Anfrage nach Abschnitt 4.6 die Länge der Anfrage zunimmt, was sich direkt in erhöhter Laufzeit niederschlägt. Eine Laufzeitanalyse des Doppelschrittindex ergab, dass sich die Laufzeit im Verhältnis $\frac{1}{7}$ zu $\frac{6}{7}$ auf die Anfragen an Knotenindex und Dokumentindex (siehe Abschnitt 4.6.3) aufteilen. Diese Beobachtung ist damit zu erklären, dass im ersten Schritt der Anfrageverarbeitung jeweils eine relativ große Menge an relevanten Knoten identifiziert wird, die im zweiten Schritt zu einer deutlichen Vergrößerung der Menge der Teilanfragen führt. Da die Laufzeit einer Lucene-Anfrage u.a. von der Anzahl der Teilanfragen abhängt, macht sich dies ebenfalls im Laufzeitverhalten des Doppelschrittindex bemerkbar.

5.6.2 Suche nach ähnlichen Dokumenten

Als letzte Messreihe in diesem Kapitel wird die Laufzeit von Suchanfragen nach ähnlichen Dokumenten analysiert. Wieder wurde die Messung anhand der Dokumente

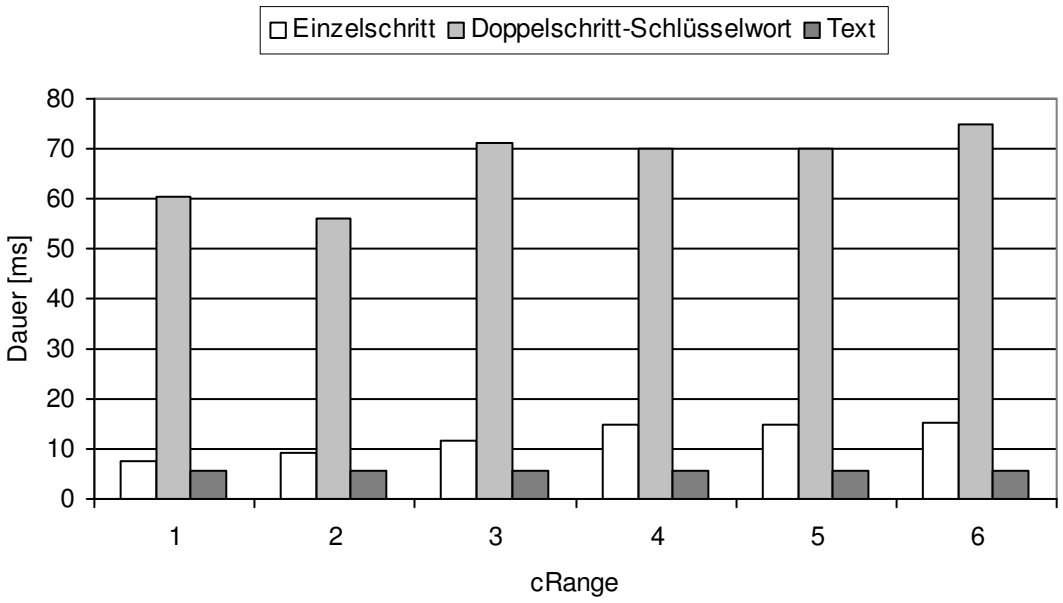


Abbildung 5.16: Dauer der Anfrageverarbeitung bei Schlüsselwortsuche. Laufzeit in Abhängigkeit von $cRange$ für den Datensatz 70k.

durchgeführt, die auch zur Bewertung der Qualität der Suchresultate verwendet wurden. Die Messwerte in Abbildung 5.17 zeigen die Anfragedauer in Abhängigkeit der Datensatzgröße für $cRange$=4. Abbildung 5.18 illustriert die Anfragedauer in Abhängigkeit von $cRange$ bei festem Datensatz 70k. Verglichen werden jeweils die in Abschnitt 5.2.3 beschriebenen Varianten der Suche nach ähnlichen Dokumenten mittels $\mathrm{ctf.icf}$ und Einzelschrittindex, mittels $\mathrm{nr.inf}$ und Doppelschrittindex für Ähnlichkeitssuche und mittels der reinen Textsuche.

Analyse der Messergebnisse Bei der Analyse der Messergebnisse aus beiden Abbildungen fällt zunächst auf, dass die Bearbeitung der Anfragen auf Basis des Einzelschrittindex wesentlich mehr Zeit benötigt als auf beiden anderen Indexen. Dies ist zunächst insofern verwunderlich, als Schlüsselwortanfrage nach Abbildung 5.15 deutlich performanter auf dem Einzelschrittindex auszuführen sind als auf dem Doppelschrittindex. Der Unterschied ist in der Länge der tatsächlich an Lucene abgesetzten Anfrage zu finden, d.h. in der Anzahl der in der Anfrage enthaltenen Teilanfragen. Während beim Einzelschrittindex alle im Präfix des Ausgangsdokuments enthaltenen Begriffe in allen L-Fields des Kontexts gesucht werden, kann diese Suche bei den beiden anderen Varianten auf das Text-L-Field beschränkt und somit auf ein einziges Feld reduziert werden; die Anfragen sind somit deutlich kürzer. Zusätzlich profitiert die Doppelschritt-Variante davon, dass nach den oben angestellten Überlegungen (siehe Abschnitt 5.2.3) für $cRange = k$ nur $\frac{k}{2}$ L-Fields durchsucht werden müssen.

Weiterhin fällt bei der Analyse der Messergebnisse nach Abbildung 5.17 die Abhängigkeit der Laufzeiten von der Größe des Datensatzes auf. Viel deutlicher als

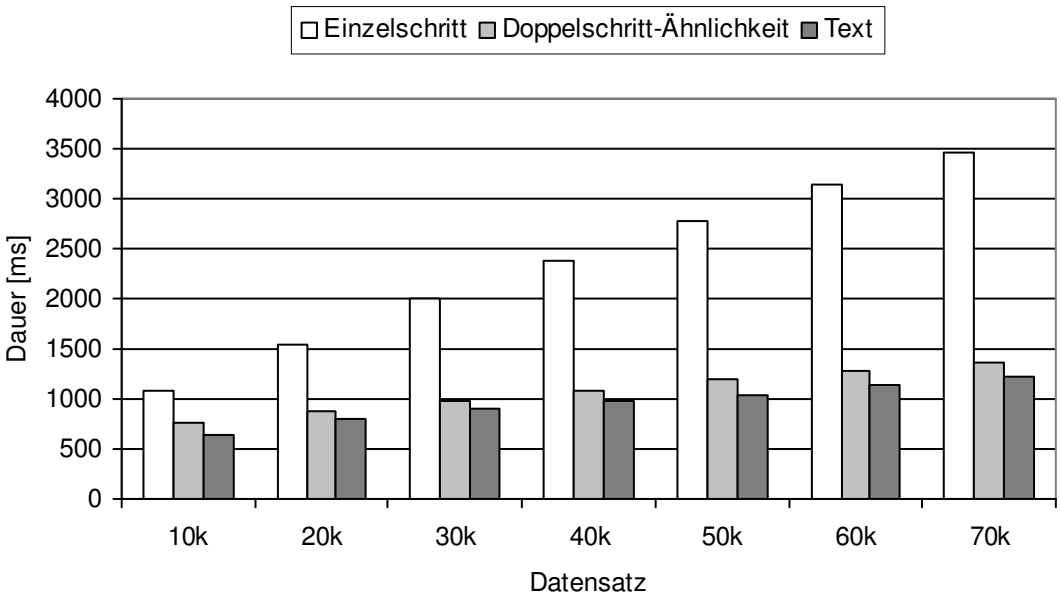


Abbildung 5.17: Dauer der Anfrageverarbeitung bei der Suche nach ähnlichen Dokumenten. Laufzeit in Abhängigkeit der Datensatzgröße für $cRange$=4.

bei der Schlüsselwortsuche (siehe Abbildung 5.15) lässt sich ein linearer Zusammenhang feststellen. Der Grund hierfür ist in der Anzahl der gefundenen Dokumente zu finden. Da die Schlüsselwortanfragen nur aus einigen wenigen Suchbegriffen bestehen, werden bei der Schlüsselwortsuche auch insgesamt weniger Dokumente gefunden. Eine Analyse der Anzahl der gefundenen Dokumente ergab, dass bei der Suche nach ähnlichen Dokumenten im Durchschnitt mehr als 99% aller Dokumente irgendeinen der Suchbegriffe enthalten und daher in die Resultatmenge aufgenommen werden, während bei der Schlüsselwortsuche dies im Durchschnitt für weniger als 1% der Fall ist.

Die beiden bis hierher diskutierten Ergebnisse der Analyse leiten sich daraus ab, dass Anfragelaufzeiten in Lucene einerseits von der Länge der Anfrage und andererseits von der Anzahl der gefundenen Dokumente abhängt. Aus der Tatsache, dass beide Parameter bei der Suche nach ähnlichen Dokumenten deutlich größer ausfallen als bei der Schlüsselwortsuche, erklärt sich das Phänomen, dass die Suche nach ähnlichen Dokumenten deutlich längere Laufzeiten aufweist. Die Laufzeiten von Doppelschrittindex und reiner Textsuche für den Datensatz 70k und $cRange$=4 liegen mit 1,35 bzw. 1,21 Sekunden knapp über der gewünschten Grenze von einer Sekunde. Der Mehraufwand von Einzelschrittindex und Doppelschrittindex verglichen mit dem reinen Textindex beträgt durchschnittlich 186% bzw. 12%.

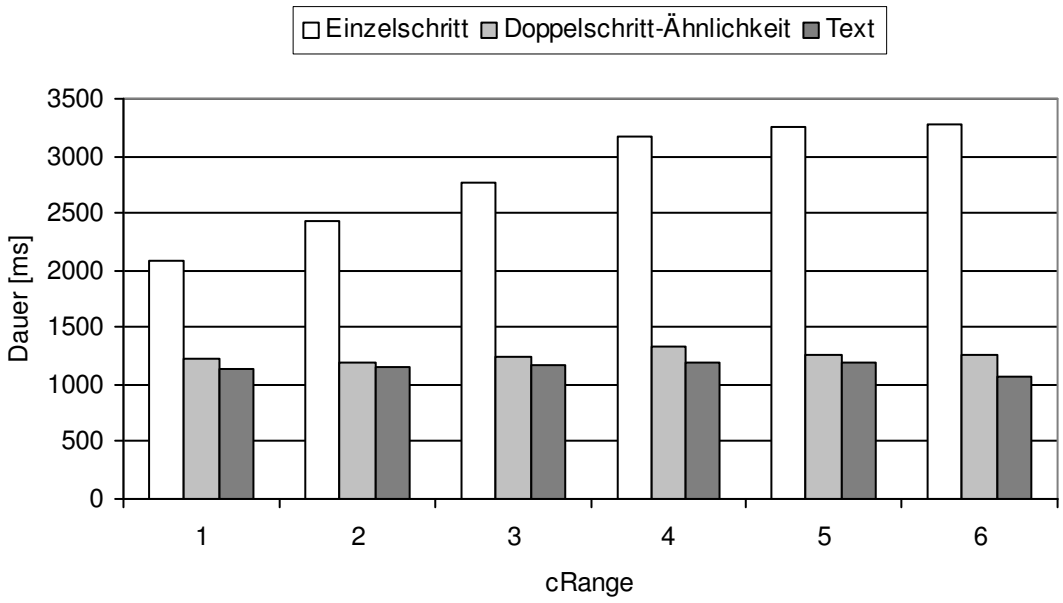


Abbildung 5.18: Dauer der Anfrageverarbeitung bei der Suche nach ähnlichen Dokumenten. Laufzeit in Abhängigkeit von $cRange$ für den Datensatz 70k.

5.7 Zusammenfassung

In den vorangegangenen Abschnitten wurde die im Rahmen der vorliegenden Arbeit implementierte Suchmaschine hinsichtlich der Qualität ihrer Suchresultate sowie hinsichtlich Indexgröße und Laufzeit analysiert. Dabei wurde zwischen der Schlüsselwortsuche und der Suche nach ähnlichen Dokumenten sowie zwischen verschiedenen Implementierungsmöglichkeiten unterschieden.

Da kein allgemeingültiges Verfahren zur Analyse und zum Vergleich von Suchmaschinen bekannt ist, wurden die Varianten der kontextbasierten Suche jeweils mit einer Referenzimplementierung der reinen Textsuche verglichen. Um die Skalierbarkeit des Systems zu analysieren, wurden die Laufzeiten und Sekundärspeicheranforderungen jeweils in Abhängigkeit von der Datensatzgröße gemessen. Wie alle Suchmaschinen verfügt auch der vorgestellte Prototyp über eine große Anzahl an Parametern, deren Belegung die Sucheigenschaften beeinflussen. Da für die kontextbasierte Suche der Parameter zur Einstellung der Kontextgröße eine zentrale Rolle spielt, wurden die Auswirkungen unterschiedlicher Belegungen des $cRange$-Parameters jeweils anhand einer Messreihe belegt. Die im folgenden genannten Zahlen beziehen sich, soweit nicht anders vermerkt, auf Messungen mit Datensatz 70k und $cRange$=4.

Die Analyse zur Qualität der Suchresultate ergab bei der kontextbasierten Schlüsselwortsuche einen deutlichen Qualitätsvorsprung vor der herkömmlichen Textsuche. Anhand des Maschinenbauszenarios konnte gezeigt werden, dass Suchanfragen, welche unter anderem Begriffe beinhalten, die nicht in Dokumenten auftauchen, klar vom Dokumentenkontext profitieren, bei ihnen wurde eine Verbesserung der Resul-

tatsqualität um 165% gemessen. Bei den anderen Suchanfragen dieses Szenarios hatte die Verwendung von Kontextinformation im Durchschnitt keine Auswirkung auf die Qualität der Resultate. Vergleichbare Anfragen im Citeseerszenario zeigten jedoch auch bei dieser Art von Anfragen einen Anstieg von 22% in der Qualität der Suchresultate gegenüber der reinen Textsuche.

Bei der Suche nach ähnlichen Dokumenten wurden die beiden kontextbasierten Bewertungsmaße $\mathrm{ctf.icf}$ und $\mathrm{nr.inf}$ mit herkömmlicher Textsuche verglichen. Wo sich für kleine Belegungen des $cRange$-Parameters bei der kontextbasierten Suche eine leichte Verbesserung der Resultate einstellte, ergab sich für größere Werte sogar eine leichte Verschlechterung. Der Nutzen der kontextbasierten Suche nach ähnlichen Dokumenten mit $\mathrm{ctf.icf}$ bzw. $\mathrm{nr.inf}$ liegt also weniger in einer Verbesserung der Suchresultate, sondern vielmehr in einer geänderten Semantik der Ähnlichkeit und somit in einer Alternative zur herkömmlichen Textsuche. Wo letztere zur Bestimmung der Ähnlichkeit von Dokumenten auf den Inhalt der Dokumente beschränkt bleibt, kann mit $\mathrm{ctf.icf}$ und $\mathrm{nr.inf}$ zusätzlich die Ähnlichkeit der Kontextinformation berücksichtigt werden. Bei der Implementierung einer Suchmaschine könnten demnach verschiedene Ähnlichkeitssuchen angeboten werden, aus denen der Benutzer je nach gewünschter Art der Ähnlichkeit auswählen kann.

Die Analyse der Algorithmen zur Berechnung der kürzesten Wege ergab einen klaren Performanz-Vorteil zugunsten des HiddenPath-Algorithmus. Wie die Analysen der Laufzeiten zur Erstellung des Indexes zeigen, wird der Hauptanteil des kontextinduzierten Mehraufwands jedoch nicht von der Berechnung der Dokumentenkontexte verursacht, sondern vom Import der Datenbankdaten.

Die Speicherplatzanforderungen und Laufzeiten zur Erstellung des Index wurden anhand dreier kontextbasierter Indexe und des reinen Textindex analysiert. Sowohl bei der Laufzeit zur Erstellung des Index als auch bei der Größe schneiden die Doppelschrittindexe deutlich besser ab als der Einzelschrittindex. Der kontextinduzierte Mehraufwand bei der Indexgröße liegt bei 11% und 19% für die Doppelschrittindexe und bei 52% für den Einzelschrittindex. Bei der Laufzeit zur Erstellung der Indexe liegt der Mehraufwand bei 71% und 72% für die Doppelschrittindexe und 119% beim Einzelschrittindex. Der bei weitem größte Anteil des Mehraufwands ist dabei dem Import der Datenbank zuzuschreiben.

Bei der Schlüsselwortsuche zeigen die Laufzeitanalysen der Suchanfragen an das System im Gegensatz zu den Analysen der Indexerstellung einen klaren Vorteil zugunsten des Einzelschrittindex. Während dieser einen gegenüber dem reinen Textindex vergleichsweise geringen Mehraufwand von 200% zeigt, zeigen die Anfragen des Doppelschrittindex mit 1400% Overhead deutlich schwächere Performanz. Selbst die Anfragebearbeitung der Doppelschrittindex-basierten Suchmaschine lag jedoch mit im Durchschnitt 60ms Laufzeit eindeutig im Bereich der akzeptablen Anfrageperformanz für interaktive Suche.

Bezüglich der Schlüsselwortsuche lässt sich nach dieser Analyse eine klassische Trade-Off-Situation zwischen Indexierungszeit und Anfragezeit feststellen. Zwar be-

nötigt der Einzelschrittindex zur Erstellung des Index mehr Zeit und mehr Speicherplatz. Dieser erhöhte Ressourcenverbrauch zur Indexierungszeit ermöglicht jedoch eine deutlich bessere Performanz zur Anfragezeit.

Diese Beobachtung gilt allerdings ausschließlich für die Bearbeitung von Schlüsselwortanfragen. Bei der Suche nach ähnlichen Dokumenten wurde festgestellt, dass aufgrund seiner Struktur der Doppelschrittindex hier bessere Performanz zeigt. Die Analyse der Suche nach ähnlichen Dokumenten ergab weiterhin, dass die große Anzahl an Suchbegriffen sich deutlich negativ auf die Performanz der Anfragen auswirkt. Daher lag sogar die durchschnittliche Dauer von Anfragen an den reinen Textindex bei knapp über einer Sekunde. Der kontextinduzierte Mehraufwand für Einzelschritt- bzw. Doppelschrittindex lag bei durchschnittlich 186% bzw. 12%.

Zusammenfassend zeigen die Messungen in Abhängigkeit der Datensatzgröße eine gute Skalierbarkeit des Systems nach Anforderung 5 in Abschnitt 1.2. Keine der Messungen enthält Hinweise darauf, dass der Ressourcenbedarf des analysierten Systems mehr als linear in der Größe des Datensatzes anwächst. Weiterhin konnte gezeigt werden, dass die Berücksichtigung von Kontext zwar einerseits mit gewissem Mehraufwand verbunden ist, jedoch andererseits zu deutlich verbesserter Resultatsqualität führt.

6

Zusammenfassung und Ausblick

Dieses Kapitel subsumiert die vorliegende Arbeit. Abschnitt 6.1 fasst zunächst einige wesentliche Merkmale der vorliegenden Arbeit zusammen. In Abschnitt 6.2 wird zur besseren Einordnung des Ansatzes anschließend eine Kategorisierung anhand des in Abschnitt 2.5.2 eingeführten Schemas vorgenommen. Die Abdeckung der zu Beginn der Arbeit eingeführten Anforderungen wird in Abschnitt 6.3 besprochen, bevor in Abschnitt 6.4 ein Ausblick auf weitere Arbeiten gegeben wird.

6.1 Zusammenfassung

Der in der vorliegenden Arbeit präsentierte Ansatz befasst sich mit kontextbasierter Dokumentensuche im Intranet. Er entstand aus der Motivation, herkömmliche, d.h. textbasierte, Dokumentensuche im Intranet von Unternehmen durch die Verwendung von Kontextinformation zu verbessern. Der hierfür notwendige Kontext wird nicht wie bei vielen Ansätzen zur semantischen Suche von Domänenexperten erstellt und in Wissensstrukturen abgelegt, sondern aus den Datenbanken des Unternehmens extrahiert. Dieses Vorgehen garantiert nicht nur die Konsistenz der Daten, sondern ebenfalls die Wichtigkeit der verwendeten Information für das Unternehmen.

Der ContextGraph dient dabei zur Abstraktion der Unternehmensdatenbanken und als Grundlage zur Bestimmung des Kontexts von Dokumenten. Darauf aufbauend wurden zur Bewertung von Suchresultaten die Bewertungsmaße $\mathrm{ctf.icf}$ und $\mathrm{nr.inf}$ entwickelt.

Der hier vorgestellte Ansatz zur kontextbasierten Suche kann in dreierlei Hinsicht als eine Erweiterung der herkömmlichen textbasierten Suche gesehen werden. Zum Ersten wurde, zur Bewertung der Suchresultate von der $\mathrm{tf.idf}$-Funktion der her-

kömmlichen Suche die kontextbasierten Bewertungsmaße $\mathrm{ctf.icf}$ und $\mathrm{nr.inf}$ abgeleitet. Zum Zweiten basiert der Prototyp in technologischer Hinsicht auf frei verfügbarer Software für die herkömmliche Suche. Auf diese Weise kann die kontextbasierte Suche leicht auf den aktuellen Stand der Technik bei herkömmlicher Suche zurückgreifen, ohne umfassende Implementierungen erforderlich zu machen. Von diesem Umstand profitiert vor allem der Ressourcenbedarf des Prototyps. Zum Dritten ist das System schließlich in der Lage, gleichzeitig sowohl Dokumente mit und ohne Kontextinformation zu verarbeiten. Nur im günstigsten Falle können im Unternehmen sämtliche Dokumente durch Datenbankdaten kontextualisiert werden. Dadurch, dass kontextbasierte Suche als eine Erweiterung herkömmlicher Suche behandelt wird, können beide Arten von Dokumenten nicht nur im Index nebeneinander existieren, sondern ebenfalls vom Benutzer im Rahmen einer einzelnen Suchanfrage gefunden werden.

Die Implementierung folgt einer komponentenbasierten Architektur. Anhand des Einzelschrittindex und der beiden Doppelschrittindexe wurden verschiedene Indexstrukturen sowie die Umsetzung der kombinierten Bewertungsmaße diskutiert. Dabei wurde unterschieden zwischen der Suche mit Schlüsselwörtern und der Suche nach ähnlichen Dokumenten. Die Analyse des Prototyps zeigte schließlich, dass sich bei geringem Overhead die Qualtiät der Suchresultate durch Kontextinformation signifikant verbessern lässt.

6.2 Einordnung des Systems

In Abschnitt 2.5.2 wurden sieben Merkmale zur Kategorisierung von Ansätzen zur semantischen Suche aufgestellt. Im Folgenden wird der im Rahmen der vorliegenen Arbeit implementierte Prototyp entlang dieser Kategorisierung diskutiert.

- Architektur. Der vorliegende Prototyp unterhält einen *eigenen Index*, in dem Informationen aus Dokumenten sowie aus deren Kontext abgespeichert werden. Es wurde kein Konzept zur Einbindung anderer Suchmaschinen entworfen.

- Kopplung. Die kontextbasierte Suche erfordert eine *enge Kopplung* zwischen Dokumenten und Datenbankinformation. Es sind jedoch durchaus semantische Erweiterungen denkbar, welche auf loser Kopplung basieren, siehe Abschnitt 4.7.

- Transparenz. Die Einstellungsmöglichkeiten der kontextbezogenen Parameter können für den unerfahrenen Benutzer leicht verborgen werden. Dem versierten Benutzer werden erweiterte Parametrisierungsmöglichkeiten geboten. Die Benutzerschnittstelle kann somit als *hybrid* klassifiziert werden.

- Benutzerkontext. Der Kontext von Benutzern wurde im Prototyp nicht weiter beachtet.

- Anfragemodifikation. Die Auswertung des Dokumentenkontexts für Suchanfragen basiert auf im ContextGraph abgelegter Information und fällt somit in die Kategorie der *graphenbasierten* Ansätze.

- Struktur der Ontologie. Im Unterschied zu den in Abschnitt 2.5 behandelten Ansätzen werden im vorliegenden Prototyp keine Ontologien eingesetzt, sondern relationale Datenbanken. Dafür kommen grundsätzlich Datenbanken aus allen Anwendungsgebieten in Betracht. Lediglich bei der Gewichtung der Kanten besteht die Möglichkeit domänenspezifische Anpassungen vorzunehmen. Der Prototyp fällt somit in die Kategorie der anonymen Strukturen, wobei fakultativ domänenspezifische Optimierungen vorgenommen werden können.

- Ontologiebeschreibungssprache. Im Protoyp liegt der Fokus nicht auf Ontologien. Mittels der in Abschnitt 4.4 besprochenen Importkomponenten können jedoch Ontologien im OWL- und RDF-Format verarbeitet werden.

6.3 Abdeckung der Anforderungen

Um den groben Rahmen des im Rahmen dieser Arbeit entwickelten Systems abzustecken, wurden in Abschnitt 1.2 einige Anforderungen formuliert. Die Erfüllung dieser Anforderungen auf die vorgestellten Konzepte und das System werden an dieser Stelle kurz zusammengefasst.

- Schlüsselwortsuche. Um eine möglichst einfache Benutzerschnittstelle anbieten zu können, wurde eine Schlüsselwortsuche implementiert. Ohne die Anforderung der Schlüsselwortsuche hätte ein System entworfen werden können, bei dem Suchanfragen als Ausdruck einer formalen Sprache definiert werden müssen.

- Transparenz. Die Anforderung nach Transparenz wirkt sich im System dahingehend aus, dass für einen unerfahrenen Benutzer keine Notwendigkeit besteht, von der Kontextbasiertheit der Suchmaschine Notiz zu nehmen.

- Anpassbarkeit und Erweiterbarkeit. In der vorliegenden Arbeit wird davon ausgegangen, dass keine Suchmaschine für Intranets alle Szenarien in allen Unternehmen abdecken kann. Um eine leichte Anpassbarkeit und Erweiterbarkeit der Maschine zu gewährleisten, wurde ein komponentenbasierter Architekturansatz gewählt.

- Textsuche. Die kontextbasierte Suche der vorliegenden Arbeit ist, wie oben diskutiert, in dreierlei Hinsicht als eine Erweiterung textbasierter Suche zu sehen. Erstens wurden die textbasierten Bewertungsmaße für Suchresultate auf das kontextbasierte Szenario erweitert. Zweitens basiert die Implementierung der Suchmaschine auf Technologien aus dem Bereich der herkömmlichen, textbasierten Suche und drittens können vom System sowohl kontextualisierte als auch nicht-kontextualisierte Dokumente erfasst und gefunden werden.

- Performanz und Skalierbarkeit. Wie in Kapitel 5 diskutiert, wächst der Ressourcenverbrauch des Systems maximal linear mit der Größe des verwendeten Datensatzes. Der im Rahmen der vorliegenden Arbeit implementierte Prototyp eignet sich gut für die interaktive Suche.

- Qualität. Bei der Schlüsselwortsuche profitiert die Qualität der Suchresultate deutlich von der Beachtung des Dokumentenkontexts, siehe Abschnitt 5.2. Bei der Suche nach ähnlichen Dokumenten steigt die Qualität gegenüber der reinen Textsuche nicht; es werden jedoch andere Dokumente gefunden, d.h. die Art der Ähnlichkeit ändert sich.

- Integration. Bereits vom Ansatz her ist der Ansatz der vorliegenden Arbeit darauf ausgelegt, Daten aus im Unternehmen bereits vorhandenen Datenbanken zu verarbeiten und zu einer Verbesserung der Dokumentensuche einzusetzen. Diese Fähigkeit hebt den Ansatz fundamental von den in Abschnitt 2.5 vorgestellten Ansätzen zur semantischen Suche ab, die auf manuell zu erstellenden und zu wartenden Ontologien basieren.

Die in Abschnitt 1.2 aufgestellten Anforderungen können somit als erfüllt betrachtet werden.

6.4 Ausblick

Basierend auf der vorgestellten Arbeit sind eine Reihe weitergehender Aktivitäten denkbar. In diesem Abschnitt wird zum Abschluss auf mögliche Erweiterungen sowie auf die Anwendung des Systems auf weitere Szenarien eingegangen.

Der in Kapitel 4 vorgestellte Prototyp wurde anhand zweier sehr unterschiedlicher *Szenarien* erprobt. Das Maschinenbauszenario enthält reale Daten eines produzierenden Unternehmens, während das Citeseerszenario auf den Daten der gleichnamigen Suchmaschine basiert. Diese Szenarien geben einen ersten Hinweis auf die Performanz der Suchmaschine und auf die Qualität ihrer Suchresultate. Weitere Szenarien und vor allem der Betrieb innerhalb eines Unternehmens wären jedoch zur weiteren Konsolidierung der Ergebnisse wünschenswert.

Dabei ist die Frage zu klären, ob die in dieser Arbeit vorgenommene Parametrisierung der Suchmaschine auch auf andere Szenarien ohne Änderung anwendbar ist.

Insbesondere ist an dieser Stelle die Kontextgröße $cRange$ zu untersuchen, jedoch auch andere Parameter, wie z.B. die Einflussfaktoren von Text- und Kontextanteil bei der Bewertung der Suchresultate, sind von Interesse.

Ebenfalls zu untersuchen wäre der Einfluss verfeinerter Kantengewichte. Im Rahmen der untersuchten Szenarien wurden Kantengewichte auf einfache Weise manuell zugewiesen. Eine differenziertere Bewertungsstrategie könnte sich positiv sowohl auf den Ressourcenbedarf als auch auf die Qualität der Suchresultate auswirken.

Zur *Berechnung der kürzesten Wege* wurden in der vorliegenden Arbeit der Algorithmus nach Dijkstra sowie der HiddenPath-Algorithmus auf das Szenario angepasst und implementiert. Weiterhin konnte gezeigt werden, dass die k-Partitheit des ContextGraphs kaum zu einer Beschleunigung der Berechnung führen wird.

Sowohl bei dem Algorithmus nach Dijkstra als auch beim HiddenPath handelt es sich um hauptspeicherbasierte Algorithmen. Für große Datenbanken ist jedoch damit zu rechnen, dass die Menge der zu betrachtenden Daten die Grenzen des Hauptspeichers übersteigt und somit die erhöhte Zugriffszeit auf Externspeicher im Algorithmus berücksichtigt werden muss.

Es existieren eine Reihe von Arbeiten zu externspeicherbasierten Kürzeste-Wege-Algorithmen [Meyer et al., 2003; Arge et al., 2004; Kumar und Schwabe, 1996]. Keine von ihnen kann jedoch alle durch die Spezifika des ContextGraph möglichen Optimierungen ausnutzen (z.B. ganzzahlige Kantengewichte oder, dass die Berechnung eines beschränkten Dokumentenkontexts in diesem Szenario ausreichend ist.)

Von der Arbeit ausgeklammert blieb der in Abschnitt 1.4 kurz angesprochene Problembereich der *Integration von Dokumenten und Datenbanken*. In Szenarien, in denen keine Verknüpfung der beiden Welten vorhanden ist, muss zum Einsatz eines wie in der vorliegenden Arbeit vorgestellten Systems eine solche Verknüpfung zunächst hergestellt werden.

Ebenfalls im Abschnitt über ausgeklammerte Problembereiche wurde der Bereich *Textanalyse* / Informationsextraktion angesprochen. Wo bei textbasierter Suche die Bedeutung des Dokumenteninhalts analysiert wird, könnte bei kontextbasierter Suche zusätzlich die Bedeutung des Dokumentenkontexts betrachtet werden. Da sich Dokumentenkontext jedoch strukturell, inhaltlich und schließlich auch durch die Bewertung der semantischen Nähe fundamental vom Text von Dokumenten unterscheidet, müssten hier zunächst neue Konzepte erarbeitet werden.

Einige Erweiterungen, die sich direkt aus der Implementierung des *Prototyps* ergeben, wurden bereits in Abschnitt 4.9 zusammengefasst. Dazu zählt die Erweiterung um gängige Funktionalitäten am Markt verfügbarer Textsuchmaschinen, die Implementierung einer Crawler-Komponente und weiterer Importkomponenten sowie die Realisierung von Funktionalitäten der semantischen Suche.

In Abschnitt 2.6 zum Wissensmanagement im Unternehmen wurde der Unterschied zwischen implizitem und explizitem Wissen angesprochen. Die in dieser Arbeit vorgestellte Lösung dient der Suche nach Dokumenten und somit dem Auffinden expliziten Wissens. Gleichermaßen ist im Unternehmen jedoch die Lokalisierung im-

pliziten Wissens von Interesse, d.h. die *Suche nach Experten* [Yimam-Seid und Kobsa, 2003; Heeren und Sihn, 2002]. Der in dieser Arbeit vorgestellte ContextGraph zur Kontextualisierung von Dokumenten ließe sich gleichermaßen zur Kontextualisierung von Experten im Unternehmen einsetzen. Es ist daher zu prüfen, inwieweit die im Rahmen der vorliegenden Arbeit entwickelten Konzepte auf die Suche nach Experten angewendet werden können.

Um die Ergebnisse von Suchmaschinen übersichtlicher zu gestalten, existieren Konzepte zur Gruppierung (engl.: *Clustering*) von Dokumenten, z.B. [Kummamuru et al., 2004; Yin et al., 2006] oder die Websuchmaschine Vivisimo[1]. Analog zur Suche kann zur Gruppierung von Dokumenten einerseits deren Inhalt und andererseits deren Kontext betrachtet werden. In diesem Zusammenhang ist zu untersuchen, inwieweit existierende Konzepte von der im ContextGraph abgelegten Information profitieren oder dahingehend erweitert werden können.

Zusammenfassend lässt sich sagen, dass der hier vorgestellte Ansatz sowie der darauf basierende Prototyp eine vielversprechende Grundlage für die kontextbasierte Dokumentensuche im Intranet bilden. Darauf aufbauend sind eine Reihe an Möglichkeiten zur Weiterentwicklung denkbar, die weit über die Dokumentensuche hinaus reichen.

[1] `http://vivisimo.com/` [04.11.2006]

Abbildungsverzeichnis

1.1 Beispielszenario. 16

2.1 Informationsfluss in indexbasierten Suchmaschinen. 27
2.2 Varianten der Anfragemodifikation. 50
2.3 Ausschnitt aus dem Stuttgarter Wissensmodell nach [Berger et al., 2004]. 59
2.4 Beschränkter HiddenPath-Algorithmus. 65
2.5 Einordnung der Ansätze bezüglich Daten und Anfragen. 66

3.1 Beispielhafte Abbildung einer RDB auf einen ContextGraph. . . . 75
3.2 Semantischer Abstand. (a) Symmetrische Definition. (b) Asymmetrische Definition. 77
3.3 Transformation T angewandt auf zwei Beispielgraphen. (a) $m > n$. (b) $n \geq m$. 83
3.4 Ein 5-partiter Graph. 85
3.5 Dokumentenknoten n_d mit $\mathrm{context}_3(n_d)$. 89
3.6 Dokumentenknoten $n_{d_1}, n_{d_2}, n_{d_3}, n_{d_4}$ mit jeweiligem Kontext. . . . 89
3.7 Beispiel zum Vektorraum-Modell mit $\mathrm{tf.idf}$. (a) Dokumente. (b) Vektoren mit $\mathrm{tf.idf}$-Gewichten nach Gleichung (3.4). 92
3.8 Beispiel zum Vektorraum-Modell mit $\mathrm{tf.idf}$. (a) Anfragen. (b) Vektoren mit $\mathrm{tf.idf}$-Gewichten nach Gleichung (3.4). 92
3.9 Beispiel zu $\mathrm{ctf.icf}$ nach Abb. 3.5. 95
3.10 Beispiel zu $\mathrm{ctf.icf}$ nach Abb. 3.6. 95
3.11 Beispiel zur UND-Verknüpfung. Anfrage A UND B mit Dokumenten d_1 und d_2. (a) Kleiner Kontext: $cRange = c_s$. Nur Dokument d_2 ist relevant. (b) Großer Kontext: $cRange = c_\ell$. Beide Dokumente d_1 und d_2 sind relevant. 98
3.12 Beispiel zur NICHT-Verknüpfung. Anfrage A UND NICHT C mit Dokumenten d_1 und d_2. (a) Kleiner Kontext: $cRange = c_s$. Dokument d_1 ist relevant. (b) Großer Kontext: $cRange = c_\ell$. Bei adaptiver Kontextgröße beeinflussen verschiedene Werte n_1, n_2, n_3 für $nRange$ die Größe der Resultatmenge. 99

3.13 (a) Dokumente d_1, d_2 und d_3 in ihrem Kontext. (b) Vektoren im $|\mathbb{T}^{\mathbb{C}}|$-dimensionalen Vektorraum zur Berechnung von $\mathrm{ctf.icf}$. (c) Vektoren im $|V|$-dimensionalen Vektorraum zur Berechnung von $\mathrm{nr.inf}$. Der Platzhalter x bezeichnet Vektoreinträge > 0. 107

4.1 Indexer. 111
4.2 Searcher. 112
4.3 Gewichtung der Kanten. 116
4.4 Beispieldaten an den Schnittstellen der Kontextualisierungskomponente. 118
4.5 Beispieldaten bei der Erstellung des Einzelschrittindex. 124
4.6 Einzelschrittindex vs. Doppelschrittindex. (a) Modifizierter Beispielgraph (b) Logischer Aufbau des Einzelschrittindex. (c) Logischer Aufbau des Doppelschrittindex bestehend aus Knotenindex und Dokumentindex. 130
4.7 Parameterbelegung beim Doppelschrittindex. 133
4.8 Benutzerschnittstelle des Prototypen. 137
4.9 Bewertung eines Dokuments. 138

5.1 ER-Diagramm des Maschinenbauszenarios. 142
5.2 ER-Diagramm des Citeseerszenarios. 143
5.3 Anzahl der kürzesten Wege. (a) Für festes $cRange$, variable Datensatzgröße. (b) Für feste Datensätze, variables $cRange$. 145
5.4 Qualität als Precision@10 von Suchresultaten im Maschinenbauszenario in Abhängigkeit von $cRange$ bzw. von Google Desktop für Kontext- und Textanfragen sowie für deren Durchschnitt. 148
5.5 Qualität als Precision@10 von Suchresultaten im Citeseerszenario in Abhängigkeit von $cRange$ bzw. von Google Desktop. 149
5.6 Verteilung der Dokumente der Resultate in Google Desktop, der Kontextsuchmaschine mit $cRange$=0 und Verteilung des Dokumentenstamms. 150
5.7 Qualität als Precision@10 bei der Suche nach ähnlichen Dokumenten. Vergleich von $\mathrm{nr.inf}$ mit $\mathrm{ctf.icf}$ in Abhängigkeit von $cRange$. 152
5.8 Indexgrößen in Abhängigkeit der Datensatzgröße für $cRange$=4. Jeweils Größe des Einzelschrittindex, des Doppelschrittindex für Schlüsselwortsuche, des Doppelschrittindex für Ähnlichkeitssuche und des reinen Textindex. 154
5.9 Indexgrößen in Abhängigkeit von $cRange$ für Datensatz 70k. . . . 155
5.10 Laufzeit zur Berechnung der kürzesten Wege in Abhängigkeit der Datensatzgröße für $cRange$=4. 156
5.11 Laufzeit des HiddenPath-Algorithmus in Abhängigkeit von $cRange$ für Datensatz 70k. 157

5.12 Laufzeit zur Indexerstellung: Einzelschrittindex (ES), Doppelschrittindex für Schlüsselwortsuche (DSS), Doppelschrittindex zur Ähnlichkeitssuche (DSÄ) und reiner Textindex (Text) in Abhängigkeit der Datensatzgröße für $cRange$=4. 158
5.13 Laufzeit zur Indexerstellung: Einzelschrittindex (ES), Doppelschrittindex für Schlüsselwortsuche (DSS), Doppelschrittindex zur Ähnlichkeitssuche (DSÄ) und reiner Textindex (Text) in Abhängigkeit von $cRange$ für Datensatz 70k. 159
5.14 Kontextinduzierter Mehraufwand der Laufzeit des Indexierungsschritts. (a) In Abhängigkeit der Datensatzgröße für $cRange$=4. (b) In Abhängigkeit von $cRange$ für Datensatz 70k. 160
5.15 Dauer der Anfrageverarbeitung bei Schlüsselwortsuche. Laufzeit in Abhängigkeit der Datensatzgröße für $cRange$=4. 161
5.16 Dauer der Anfrageverarbeitung bei Schlüsselwortsuche. Laufzeit in Abhängigkeit von $cRange$ für den Datensatz 70k. 162
5.17 Dauer der Anfrageverarbeitung bei der Suche nach ähnlichen Dokumenten. Laufzeit in Abhängigkeit der Datensatzgröße für $cRange$=4. 163
5.18 Dauer der Anfrageverarbeitung bei der Suche nach ähnlichen Dokumenten. Laufzeit in Abhängigkeit von $cRange$ für den Datensatz 70k. 164

Tabellenverzeichnis

2.1 Ansätze zur semantischen Suche im Vergleich. 55

3.1 Beispiel zum Vektorraum-Modell mit $\mathrm{tf.idf}$. Relevanzbewertung der Dokumente aus Abb. 3.7 bzgl. der Anfragen aus Abb. 3.8 nach Gleichung (3.3). 92
3.2 Vergleich der Funktionsdefinitionen: $\mathrm{tf}()$ vs. $\mathrm{ctf}()$ und $\mathrm{idf}()$ vs. $\mathrm{icf}()$. 94
3.3 Vergleich der Funktionsdefinitionen: $\mathrm{tf}()$ vs. $\mathrm{ctf}()$ und $\mathrm{nr}()$; $\mathrm{idf}()$ vs. $\mathrm{icf}()$ und $\mathrm{inf}()$. 106

5.1 Charakteristika der Datensätze des Citeseer-Szenarios. 144
5.2 Größe des Einzelschrittindex vs. Größe des Dokumentenstamms. . 154

Literaturverzeichnis

Abiteboul, S., Buneman, P., und Suciu, D. (2000). *Data on the Web - From relations to semistructured data and XML.* Morgan Kaufmann Publishers.

Abiteboul, S., Quass, D., McHugh, J., Widom, J., und Wiener, J. L. (1997). The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88.

Agrawal, S., Chaudhuri, S., und Das, G. (2002). DBXplorer: A system for keyword-based search over relational databases. In *ICDE'02: Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, Seite 5.

Al-Khalifa, S., Yu, C., und Jagadish, H. V. (2003). Querying structured text in an XML database. In *SIGMOD'03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, Seiten 4–15, New York, NY, USA. ACM Press.

Amaral, C., Laurent, D., Martins, A., Mendes, A., und Pinto, C. (2004). Design and implementation of a semantic search engine for Portuguese. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004), Volume 1*, Seiten 247–250.

Amer-Yahia, S., Koudas, N., Marian, A., Srivastava, D., und Toman, D. (2005). Structure and content scoring for XML. In *VLDB'05: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Seiten 361–372. VLDB Endowment.

Amer-Yahia, S. und Shanmugasundaram, J. (2005). XML full-text search: Challenges and opportunities. In *VLDB'05: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Seiten 1368–1368. VLDB Endowment. Folien online verfügbar unter: http://www.vldb2005.org/program/slides/fri/s1368-amer-yahia.ppt [10.11.2006].

Arge, L., Meyer, U., und Toma, L. (2004). External memory algorithms for diameter and all-pairs shortest-paths on sparse graphs. In *Automata, Languages and*

Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings, Seiten 146–157. Springer.

Baader, F., Calvanese, D., McGuiness, D. L., Nardi, D., und Patel-Schneider, P. F., Hrsg. (2003). *The description logics handbook*. Cambridge University Press.

Baeza-Yates, R. und Ribeiro-Neto, B. (1999). *Modern information retrieval*. ACM press.

Bai, J., Song, D., Bruza, P., Nie, J.-Y., und Cao, G. (2005). Query expansion using term relationships in language models for information retrieval. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 688–695, New York, NY, USA. ACM Press.

Balmin, A., Hristidis, V., und Papakonstantinou, Y. (2004). ObjectRank: Authority-based keyword search in databases. In *VLDB'04: Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3, 2004*, Seiten 564–575.

Banks, D., Cayzer, S., Dickinson, I., und Reynolds, D. (2002). The ePerson snippet manager: A Semantic Web application. Technischer Bericht HPL-2002-328, Digital Media Systems Laboratory HP Laboratories Bristol.

Bauer, A. und Günzel, H., Hrsg. (2004). *Data Warehouse Systeme*. dpunkt.verlag.

Berger, S., Mangold, C., und Meyer, S. (2004). Wissensmanagement für die wandlungsfähige Montage. *wt Werkstattstechnik*, 3:80–85. Springer VDI Verlag, ISSN: 1436-4980.

Berger, S., Mangold, C., und Meyer, S. (2005a). Ontologiebasiertes Wissensmanagement in der Montage - Wissen in turbulenten Zeiten strukturiert einsetzen. *Industrie Management*, 21(1):49–52. Gito Verlag, Berlin, ISSN: 1434-1980.

Berger, S., Mangold, C., Meyer, S., und Westkämper, E. (2005b). Knowledge management in assembly. In *CIRP: 38th International Seminar on Manufacturing Systems*. 7 Seiten.

Berners-Lee, T., Hendler, J., und Lassila, O. (2001). The semantic Web. *Scientific American*, 2001(5).

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., und Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *ICDE'02: Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, Seite 431.

Bizer, C. und Seaborne, A. (2004). D2RQ – Treating non-RDF databases as virtual RDF graphs. In *3rd International Semantic Web Conference (ISWC2004)*. 2 Seiten (Poster).

Böhlen, M., Bukauskas, L., und Dyreson, C. (1999). The Jungle database search engine. In *ICDE'98: Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, Seiten 584–586. ACM Press.

Brassard, G. und Bratley, P. (1988). *Algorithmics – Theory and practice*. Prentice Hall.

Brickley, D. und Guha, R. (2004). *RDF vocabulary description language 1.0: RDF schema – W3C recommendation 10 February 2004*. W3C. Online verfügbar: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ [17.01.2006].

Brin, S. und Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. In *WWW'98: Proceedings of the Seventh International World Wide Web Conference*, Seiten 107–117.

Bry, F., Kraus, M., Olteanu, D., und Schaffert, S. (2001). Aktuelles Schlagwort: Semistrukturierte Daten. *Informatik-Spektrum*, 24(4):230–233.

Budanitsky, A. (1999). Lexical semantic relatedness and its application in natural language processing. Technischer Bericht CSRG-390, Department of Computer Science, University of Toronto. Online verfügbar: http://www.cs.toronto.edu/compling/Publications/Abstracts/Theses/Budanitsky-thabs.html [03.02.2006].

Budanitsky, A. und Hirst, G. (2001). Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources, in the North American Chapter of the Association for Computational Linguistics (NAACL-2001)*. 6 Seiten.

Buehler, B. (2005). *Intro to WebSphere Information Integrator OmniFind edition*. IBM. Online verfügbar: https://www6.software.ibm.com/dw/education/dm/dm0503buehler/ [07.11.2006].

Buneman, P., Davidson, S. B., Hillebrand, G. G., und Suciu, D. (1996). A query language and optimization techniques for unstructured data. In *SIGMOD'96: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data 1996, Montreal, Quebec, Canada, June 04 - 06, 1996*, Seiten 505–516. ACM Press.

Burton-Jones, A., Storey, V. C., Sugumaran, V., und Purao, S. (2003). A heuristic-based methodology for semantic augmentation of user queries on the Web. In

Proceedings of the 22th International Conference on Conceptual Modeling, ER 2003, Seiten 476–489. Springer.

Busse, S., Kutsche, R.-D., Leser, U., und Weber, H. (1999). Federated information systems: Concepts, terminology and architectures. Technischer Bericht, Technische Universität Berlin.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., und Rosati, R. (2004). What to ask to a peer: Ontology-based query reformulation. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*, Seiten 469–478.

Cencini, A. (2006). *SQL Server 2005 full-text search: Internals and enhancements*. Online verfügbar http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/sql2005ftsearch.asp [14.11.2006].

Chakaravarthy, V. T., Gupta, H., Roy, P., und Mohania, M. K. (2006). Efficiently linking text documents with relevant structured information. In *VLDB'06: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, Seiten 667–678.

Chaudhuri, S. und Dayal, U. (1997). An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74.

Chaudhuri, S., Ramakrishnan, R., und Weikum, G. (2005). Integrating DB and IR technologies: What is the sound of one hand clapping? In *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005*, Seiten 1–12.

Choy, D. M. (2005). Integration of structured and unstructured data in IBM Content Manager. In *SIGMOD'05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, Seiten 811–816, New York, NY, USA. ACM Press.

Ciravegna, F., Dingli, A., Petrelli, D., und Wilks, Y. (2002). User-system cooperation in document annotation based on information extraction. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, Seiten 122–137. Springer-Verlag.

Cody, W. F., Kreulen, J. T., Krishna, V., und Spangler, W. S. (2002). The integration of business intelligence and knowledge management. *IBM Systems Journal*, 41(4):697–713.

Cohen, S., Mamou, J., Kanza, Y., und Sagiv, Y. (2003). XSEarch: A semantic search engine for XML. In *VLDB'03: Proceedings of the 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, Seiten 45–56.

Collins, A. M. und Loftus, E. F. (1975). A spreading activation theory of semantic processing. *Psychological Review*, 82(6):407–428.

Computerwoche (2006). Eine Suchmaschine für Netweaver. *Computerwoche*, 2006(9):1.

Conrad, S. (1997). *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag Berlin Heidelberg New York.

Cormen, T. H., Leiserson, C. E., und Rivest, R. L. (1994). *Introduction to algorithms*. MIT Press.

Crestani, F. (1997). Application of spreading activation techniques in information retrieval. *Artificial Intelligence Reviews*, 11(6):453–482.

Crestani, F. und Lee, P. L. (1999). WebSCSA: Web search by constrained spreading activation. In *Proceedings of IEEE ADL 99 - Advances in Digital Libraries Conference*, Seiten 163–170.

Cyganiak, R. und Bizer, C. (2006). D2R Server: A Semantic Web front-end to existing relational databases. In *XML Tage 2006*, Seiten 171–173.

Dar, S., Entin, G., Geva, S., und Palmon, E. (1998). DTL's DataSpot: Database exploration using plain language. In *VLDB'98: Proceedings of the 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Seiten 645–649. Morgan Kaufmann.

Date, C. J. (1982). A formal definition of the relational model. *SIGMOD Record*, 13(1):18–29.

Davies, J., Weeks, R., und Krohn, U. (2004). QuizRDF: Search technology for the Semantic Web. In *HICSS '04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4*, Seiten 112–119, Washington, DC, USA. IEEE Computer Society.

Dean, J. und Henzinger, M. R. (1999). Finding related pages in the World Wide Web. In *WWW'99: Proceedings of the Eighth International World Wide Web Conference*, Seiten 1467–1479. Elsevier North-Holland, Inc.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., und Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science (JASIS)*, 41(6):391–407.

Dessloch, S. und Mattos, N. (1997). Integrating SQL databases with content-specific search engines. In *VLDB'97: Proceedings of the 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, Seiten 528–537, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ding, H., Solvberg, I. T., und Lin, Y. (2004). A vision on semantic retrieval in P2P network. In *Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)*, Seiten 177– 182.

Ehrig, M., Tempich, C., Broekstra, J., van Harmelen, F., Sabou, M., Siebes, R., Staab, S., und Stuckenschmidt, H. (2003). SWAP: Ontology-based Knowledge Management with Peer-to-Peer Technology. In *Proceedings of the second Konferenz Professionelles Wissensmanagement*, Seite 4 Seiten, Lucern.

Euzenat, J. (2004). State of the art on ontology anlignment. Technischer Bericht, INRIA.

Fellbaum, C., Hrsg. (1998). *Wordnet: an electronic lexical database*. MIT Press.

Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., und Suciu, D. (1998). Catching the boat with Strudel: Experiences with a Web-site management system. In *SIGMOD'98: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data 1998, Seattle, Washington, United States, June 01 - 04, 1998*, Seiten 414–425. ACM Press.

Finin, T., Mayfield, J., Joshi, A., Cost, R. S., und Fink, C. (2005). Information retrieval and the Semantic Web. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. 10 Seiten.

García, E. und Sicilia, M.-Á. (2003). User interface tactics in ontology-based information seeking. *PsychNology Journal*, 1(3):242–255.

Glover, E. J., Lawrence, S., Gordon, M. D., Birmingham, W. P., und Giles, C. L. (2001). Web search – your way. *Communications of the ACM*, 44(12):97–102.

Goldman, R., Shivakumar, N., Venkatasubramanian, S., und Garcia-Molina, H. (1998). Proximity search in databases. In *VLDB'98: Proceedings of the 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Seiten 26–37. Morgan Kaufmann Publishers Inc.

Goldman, R. und Widom, J. (2000). WSQ/DSQ: a practical approach for combined querying of databases and the Web. In *SIGMOD'00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data 2000, Dallas, Texas, United States, May 15 - 18, 2000*, Seiten 285–296. ACM Press.

Google (2003). Simplicity and enterprise search – A new model for managing your enterprise information. White paper. Online verfügbar http://www.google.com/enterprise/pdf/google_simplicity_enterprise_wp.pdf [13.11.2006].

Gospodnetić, O. und Hatcher, E. (2005). *Lucene in action*. Manning Publications.

Gottschick, J., Friedrich, H., und Billig, A. (2005). Marktübersicht Wissensportale. *Informatik Spektrum*, 28(3):202–209.

Graupmann, J., Schenkel, R., und Weikum, G. (2005). The SphereSearch engine for unified ranked retrieval of heterogeneous XML and Web documents. In *VLDB'05: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Seiten 529–540. VLDB Endowment.

Grossman, D. A. und Frieder, O. (2004). *Information retrieval algorithms and heuristics*. The Information Retrieval Series, Vol. 15. Springer, 2. Auflage. ISBN: 1-4020-3004-5.

Grossman, D. A., Frieder, O., Holmes, D. O., und Roberts, D. C. (1997). Integrating structured data and text: A relational approach. *Journal of the American Society for Information Science (JASIS)*, 48(2):122–132.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

Guarino, N., Masolo, C., und Vetere, G. (1999). OntoSeek: Content-based access to the Web. *Intelligent Systems and their Applications, IEEE*, 14(3):70 – 80.

Guha, R., McCool, R., und Miller, E. (2003). Semantic search. In *WWW'03: Proceedings of the 12th International Conference on World Wide Web 2003, Budapest, Hungary, May 20 - 24, 2003*, Seiten 700–709, New York, NY, USA. ACM Press.

Guo, L., Shao, F., Botev, C., und Shanmugasundaram, J. (2003). XRANK: Ranked keyword search over XML documents. In *SIGMOD'03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, Seiten 16–27, New York, NY, USA. ACM Press.

Haveliwala, T. H. (2002). Topic-sensitive PageRank. In *WWW'02: Proceedings of the 11th International Conference on World Wide Web 2002, Honolulu, Hawaii, USA, May 07 - 11, 2002*, Seiten 517–526.

Heeren, F. und Sihn, W. (2002). Xpertfinder – Message analysis for the recommendation of contact persons within defined topics. In *Africon Conference in Africa, 2002. IEEE AFRICON. 6th*, Jgg. 1, Seiten 41– 46.

Heflin, J. und Hendler, J. (2000). Searching the Web with SHOE. In *AAAI-2000: In Artificial Intelligence for Web Search. Papers from the AAAI Workshop*, Seiten 35–40.

Helfat, C. E. und Raubitschek, R. S. (2000). Product sequencing: Co-evolution of knowledge, capabilities and products. *Strategic Management Journal*, 21:961–979.

Henderson, R. M. und Clark, K. B. (1990). Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms – Technology, organizations, and innovation. *Administrative Science Quarterly*, 35(1):9–30.

Horrocks, I. (2002). DAML+OIL: A reason-able Web ontology language. In *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings*, Jgg. 2287 aus *Lecture Notes in Computer Science*, Seiten 2–13. Springer.

Hristidis, V., Gravano, L., und Papakonstantinou, Y. (2003a). Efficient IR-style keyword search over relational databases. In *VLDB'03: Proceedings of the 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, Seiten 850–861.

Hristidis, V. und Papakonstantinou, Y. (2002). DISCOVER: Keyword search in relational databases. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, Seiten 670–681.

Hristidis, V., Papakonstantinou, Y., und Balmin, A. (2003b). Keyword proximity search on XML graphs. In *ICDE'03: Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, Seiten 367–379, Los Alamitos, CA, USA. IEEE Computer Society.

Huber, H. (2005). Selbstlernende Suche. *Informatik Spektrum*, 28(3):189–192.

Hulgeri, A., Bhalotia, G., Nakhe, C., Chakrabarti, S., und Sudarshan, S. (2001). Keyword search in databases. *IEEE Data Engineering Bulletin*, 24(3):22–32.

Hyvönen, E., Saarela, S., und Viljanen, K. (2003). Ontogator: Combining view- and ontology-based search with semantic browsing. In *Proceedings of XML Finland 2003, Open Standards, XML, and the Public Sector*. 16 Seiten.

IBM (2004). *Net Search Extender – Verwaltung und Benutzerhandbuch*. IBM Corporation. Online verfügbar: http://www-306.ibm.com/software/data/db2/extenders/netsearch/library.html [05.01.2006].

IBM (2006). *The DB2 Net Search Extender version 8.1*. IBM Corporation. Online verfügbar: ftp://ftp.software.ibm.com/software/data/db2/extenders/netsearch/db2nsespecsheet.pdf [05.01.2006].

INEX (2006). INitiative for the Evaluation of XML Retrieval. WWW-Seite. Online verfügbar: http://inex.is.informatik.uni-duisburg.de/ [18.01.2006].

Inmon, W. H. (1996). *Building the data warehouse*. Wiley.

Jansen, B. J., Spink, A., Bateman, J., und Saracevic, T. (1998). Real life information retrieval: a study of user queries on the Web. *SIGIR Forum*, 32(1):5–17.

Jeh, G. und Widom, J. (2002). SimRank: a measure of structural-context similarity. In *KDD '02: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seiten 538–543, New York, NY, USA. ACM Press.

Jelmini, C. und Marchand-Maillet, S. (2004). Ontology reasoning for multimedia semantic retrieval. In *Multimodal Interaction and Related Machine Learning Algorithms Workshop, Martigny, Switzerland*. Extended Abstract.

Karger, D. R., Koller, D., und Phillips, S. J. (1993). Finding the hidden path: Time bounds for all-pairs shortest path. *SIAM Journal on Computing*, 22(6):1199–1217.

Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., und Scholl, M. (2002). RQL: A declarative query language for RDF. In *WWW'02: Proceedings of the 11th International Conference on World Wide Web 2002, Honolulu, Hawaii, USA, May 07 - 11, 2002*, Seiten 592–603.

Kaushik, R., Krishnamurthy, R., Naughton, J. F., und Ramakrishnan, R. (2004). On the integration of structure indexes and inverted lists. In *SIGMOD'04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, Seiten 779–790, New York, NY, USA. ACM Press.

Kemper, A. und Eickler, A. (2004). *Datenbanksysteme*. Oldenbourg Wissenschaftsverlag, 5. Auflage.

Kessler, M. M. (1963). Bibliographic coupling between scientific papers. *American Documentation*, 14(1):10–25.

Khan, L., McLeod, D., und Hovy, E. H. (2004). Retrieval effectiveness of an ontology-based model for information selection. *VLDB Journal*, 13(1):71–85.

Kifer, M., Lausen, G., und Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843.

Kumar, V. und Schwabe, E. J. (1996). Improved algorithms and data structures for solving graph problems in external memory. In *SPDP '96: Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP '96)*, Seite 169. IEEE Computer Society.

Kummamuru, K., Lotlikar, R., Roy, S., Singal, K., und Krishnapuram, R. (2004). A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW'04: Proceedings of the 13th International Conference on World Wide Web 2004, New York, NY, USA, May 17-20, 2004*, Seiten 658–665, New York, NY, USA. ACM Press.

Levesque, H. J. und Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93.

Liu, S., Yu, C., und Meng, W. (2005). Word sense disambiguation in queries. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 525–532, New York, NY, USA. ACM Press.

Maedche, A. und Motik, B. (2003). Repräsentations- und Anfragesprachen für Ontologien – eine Übersicht. *Datenbank Spektrum*, 2003(6):43–53.

Mandreoli, F., Martoglia, R., und Ronchetti, E. (2005). Versatile structural disambiguation for semantic-aware applications. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 209–216, New York, NY, USA. ACM Press.

Mangold, C. (2007). A survey and classification of semantic search approaches. *International Journal of Metadata, Semantics and Ontologies (IJMSO)*, 2(1):23–34.

Mangold, C. und Mitschang, B. (2003). Enabling a reuse oriented engineering methodology. In *Information and Knowledge Sharing (IKS 2003)*, Seiten 6–11. ActaPress.

Mangold, C., Rantzau, R., und Mitschang, B. (2003). Föderal: Management of engineering data using a semistructured data model. In *ICEIS 2003, Proceedings of the 5th International Conference on Enterprise Information Systems, Angers, France, April 22-26, 2003*, Seiten 382–389.

Mangold, C. und Schwarz, H. (2006). Documents meet databases: A system for intranet search. In *13th International Conference on Management of Data (COMAD 2006), December 14-16, 2006, Delhi, India*, Seiten 201–204.

Mangold, C., Schwarz, H., und Mitschang, B. (2005). Improving intranet search-engines using context information from databases. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 349–350, New York, NY, USA. ACM Press.

Mangold, C., Schwarz, H., und Mitschang, B. (2006a). Symbiosis in the intranet: How document retrieval benefits from database information. In *13th International Conference on Management of Data (COMAD 2006), December 14-16, 2006, Delhi, India*, Seiten 227–230.

Mangold, C., Schwarz, H., und Mitschang, B. (2006b). u38: A framework for database-supported enterprise document-retrieval. In *10th International Database Engineering and Applications Symposium (IDEAS 2006), 11-14 December 2006, Delhi, India*, Seiten 193–200. IEEE Computer Society.

Manning, C. D. und Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Manola, F. und Miller, E. (2004). *RDF primer, W3C Recommendation 10 February 2004*. W3C. Online verfügbar: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ [17.01.2006].

McGuiness, D. und van Harmelen, F. (2004). *OWL web ontology language: Overview – W3C recommendation 10 February 2004*. W3C. Online verfügbar: http://www.w3.org/TR/2004/REC-owl-features-20040210/ [17.01.2006].

Meyer, S. (2007). *Wandlungsfähigkeit durch Wissensmanagement: Konzept zum Management wissensbasierter Wandlungsfähigkeit in Unternehmen*. Dissertation, Universität Stuttgart, Betriebswirtschaftliches Institut, Lehrstuhl für Planung.

Meyer, U., Sanders, P., und Sibeyn, J., Hrsg. (2003). *Algorithms for memory hierarchies - Advanced lectures*, Jgg. 2625 aus *Lecture Notes in Computer Science*. Springer.

Mihajlović, V., Blok, H. E., Hiemstra, D., und Apers, P. M. G. (2005). Score region algebra: Building a transparent XML-R database. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 12–19, New York, NY, USA. ACM Press.

Mishne, G., Carmel, D., Hoory, R., Roytman, A., und Soffer, A. (2005). Automatic analysis of call-center conversations. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 453–459, New York, NY, USA. ACM Press.

Mitra, M. und Chaudhuri, B. B. (2000). Information retrieval from documents: A survey. *Information Retrieval*, 2(2-3):141–163.

Mitra, M., Singhal, A., und Buckley, C. (1998). Improving automatic query expansion. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seiten 206–214, New York, NY, USA. ACM Press.

Moench, E., Ullrich, M., Schnurr, H.-P., und Angele, J. (2003). SemanticMiner – Ontology-based knowledge retrieval. Technischer Bericht, ontoprise Gmbh, Online verfügbar: http://www.ontoprise.de/documents/SemanticMiner_Ontology-Based_Knowledge_Retrieval.pdf [16.07.2004].

Noy, N. F. und Musen, M. A. (2000). PROMPT: Algorithm and tool for automated ontology merging and alignement. In *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Seiten 450 – 455.

Oracle (2004). *Oracle Ultra Search for 10g database release 1 technical white paper*. Oracle Corporation. Online verfügbar: http://www.oracle.com/technology/products/ultrasearch/index.html [05.01.2006].

Oracle (2005). *Oracle Text: features overview*. Oracle Corporation. Online verfügbar: http://www.oracle.com/technology/products/text/index.html [05.01.2006].

Oracle (2006). *Oracle Ultra Search for Oracle Collaboration Suite 10g data sheet*. Oracle Corporation. Online verfügbar: http://www.oracle.com/technology/products/ultrasearch/index.html [05.01.2006].

Palmon, E. (1998). Associative search method for heterogeneous databases with an integration mechanism configured to combine schema-free data models such as a hyperbase. United States Patent Number 5,740,421, 1998-05-14. Online verfügbar: http://www.uspto.gov [09.01.2006].

Palmon, E. und Geva, S. (1998). Associative search method with navigation for heterogeneous databases including an integration mechanism configured to combine schema-free data models such as a hyperbase. United States Patent Number 5,819,264, 1998-10-06. Online verfügbar: http://www.uspto.gov [09.01.2006].

Papakonstantinou, Y., Garcia-Molina, H., und Widom, J. (1995). Object exchange across heterogeneous information sources. In *ICDE'95: Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, Seiten 251–260. IEEE Computer Society.

Papineni, K. (2001). Why inverse document frequency? In *NAACL 2001, Language Technologies 2001: The Second Meeting of the North American Chapter of the Association for Computational Linguistics*, Seiten 1–8.

Pinto, H. S. und Martins, J. P. (2001). A methodology for ontology integration. In *Proceedings of the International Conference on Knowledge Capture*, Seiten 131–138. ACM Press.

Polanyi, M. (1985). *Implizites Wissen*. Suhrkamp.

Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., und Widom, J. (1995). Querying semistructured heterogeneous information. In Ling, T. W., Mendelzon, A. O., und Vieille, L., Hrsg., *Deductive and Object-Oriented Databases, Fourth International Conference, DOOD'95*, Jgg. 1013 aus *Lecture Notes in Computer Science*, Seiten 319–344. Springer.

Robertson, S. E. und Jones, K. S. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146.

Robertson, S. E., Walker, S., und Hancock-Beaulieu, M. (1998). Okapi at TREC-7: Automatic ad hoc, filtering, VLC and interactive. In *The Seventh Text REtrieval Conference (TREC-7)*, Seiten 199–210.

Rocha, C., Schwabe, D., und de Aragão, M. P. (2004). A hybrid approach for searching in the semantic Web. In *WWW'04: Proceedings of the 13th International Conference on World Wide Web 2004, New York, NY, USA, May 17-20, 2004*, Seiten 374–383.

Roy, P., Mohania, M. K., Bamba, B., und Raman, S. (2005). Towards automatic association of relevant unstructured content with structured query results. In *CIKM'05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Seiten 405–412, New York, NY, USA. ACM Press.

Salton, G. (1971). *The SMART retrieval system : experiments in automatic document processing*. Englewood Cliffs: Prentice Hall.

Salton, G. und Buckley, C. (1987). Term weighting approaches in automatic text retrieval. Technischer Bericht 87-881, Department of Computer Science, Cornell University. Ebenfalls in: Information Processing & Management, 24(5), S. 513-523, 1988.

Salton, G. und Buckley, C. (1988). On the use of spreading activation methods in automatic information retrieval. In *SIGIR'88, Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Grenoble, France, June 13-15, 1988*, Seiten 147–160. ACM.

Salton, G. und Lesk, M. E. (1968). Computer evaluation of indexing and text processing. *Journal of the ACM*, 15(1):8 – 36.

Sandkuhl, K. (2005). Wissensportale. *Informatik Spektrum*, 28(3):193–201.

Schmidt, A., Kersten, M., und Windhouwer, M. (2001). Querying XML documents made easy: nearest concept queries. In *ICDE'01: Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, Seiten 321–330, Washington, DC, USA. IEEE Computer Society.

Schroeder, R. G., Bates, K. A., und Junttila, M. A. (2002). A resource-based view of manufacturing strategy and the relationship to manufacturing performance. *Strategic Management Journal*, 23(2):105–117.

Seaborne, A. (2004). RDQL - A query language for RDF – W3C member submission 9 January 2004. Online verfügbar: http://www.w3.org/Submission/RDQL/ [15.07.2004].

Semagix (2005). Freedom white paper. Online verfügbar mit Datum vom 11.07.2005: http://www.semagix.com/documents/SemagixFreedomWhitePaperUKV4_000.pdf [05.07.2006].

Shah, U., Finin, T., Joshi, A., Cost, R. S., und Mayfield, J. (2002). Information retrieval on the semantic Web. In *CIKM'02: Proceedings of the Eleventh International Conference on Information and Knowledge Management*, Seiten 461–468. ACM Press.

Sheth, A., Bertram, C., Avant, D., Hammond, B., Kochut, K., und Warke, Y. (2002). Managing semantic content for the Web. *IEEE Internet Computing*, 6(4):80–87.

Sheth, A. P. und Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183 – 236.

Singhal, A. (2001). Modern information retrieval: A brief overview. *Bulletin of the Technical Committee on Data Engineering*, 24(4):35–44.

Sinha, V. und Karger, D. R. (2005). Magnet: Supporting navigation in semistructured data environments. In *SIGMOD'05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, Seiten 97–106, New York, NY, USA. ACM Press.

Small, H. (1973). Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269.

Soo, W. V., Yang, Y. S., Chen, L. S., und Fu, T. Y. (2004). Ontology acquisition and semantic retrieval from semantic annotated chinese poetry. In *Proceedings of the 2004 joint ACM/IEEE Conference on Digital Libraries*, Seiten 345–346. ACM Press.

Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21. auch in Vol. 60, S. 493-502, 2004.

Spärck Jones, K., Walker, S., und Robertson, S. E. (2000). A probabilistic model of information retrieval: Development and comparative experiments, Part 1 and 2. *Information Processing & Management*, 36(6):779–840.

Stillings, N. A., Feinstein, M. H., Garfield, J. L., Rissland, E. L., Rosenbaum, D. A., Weisler, S. E., und Baker-Ward, L. (1987). *Cognitive science: An introduction*. MIT Press, Cambridge, MA.

Stojanovic, N. (2003). On analysing query ambiguity for query refinement: The Librarian Agent approach. In *Proceedings of the 22th International Conference on Conceptual Modeling, ER 2003*, Seiten 490–505. Springer.

TREC (2006). Text REtrieval Conference. WWW-Seite. Online verfügbar: http://trec.nist.gov/ [18.01.2006].

W3C (1999). XML path language (XPath) Version 1.0 W3C recommendation 16 November 1999. Online verfügbar: http://www.w3.org/TR/1999/REC-xpath-19991116 [18.01.2006].

W3C (2006). XQuery 1.0: An XML query language – W3C Proposed Recommendation 21 November 2006. Online verfügbar: http://www.w3.org/TR/xquery/ [20.12.2006].

Witten, I. H., Moffat, A., und Bell, T. C. (1999). *Managing gigabytes*. Morgan Kaufmann, 2. Auflage.

Wolfram, D. (2000). A query-level examination of end user searching behaviour on the Excite search engine. In *CAIS 2000: Dimensions of a Global Information Science, Proceedings of the 28th Annual Conference*. Canadian Association for Information Science.

Yimam-Seid, D. und Kobsa, A. (2003). Expert-finding systems for organizations: problem and domain analysis and the DEMOIR approach. *Journal of Organizational Computing and Electronic Commerce*, 13(1):1–24.

Yin, X., Han, J., und Yu, P. S. (2006). LinkClus: efficient clustering via heterogeneous semantic links. In *VLDB'06: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, Seiten 427–438. VLDB Endowment.