Fuzzy-Suchmethoden
im
Information-Retrieval

Ingo R. Homann

19. Februar 2004

Fuzzy-Suchmethoden im Information-Retrieval

Dissertation

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

Ingo R. Homann

Technische Fakultät

Universität Bielefeld

19. Februar 2004

Gedruckt auf alterungsbeständigem Papier nach DIN-ISO 9706

Danksagung

Diese Dissertation entstand während meiner Arbeit als wissenschaftlicher Mitarbeiter der Arbeitsgruppe „Technische Informatik” an der Technischen Fakultät der Universität Bielefeld im Rahmen des durch die DFG finanzierten Projektes „Rechercheassistent”. An erster Stelle gilt mein Dank meinem Erstgutachter und Betreuer Prof. Alois C. Knoll sowie Prof. Franz Kummert für die freundliche Übernahme des Zweitgutachtens.

Bedanken möchte ich mich auch bei allen Kollegen und Mitarbeitern aus der Arbeitsgruppe „Technische Informatik”, insbesondere bei Ingo Glöckner für seine konstruktiven Anregungen und seinen Beistand bei inhaltlichen und organisatorischen Fragen, Thorsten Scherer für seine kritischen Kommentare und Frank Roeben, der bei Server- und Netzwerk-Problemen ein zuverlässiger Ansprechpartner war. Für das Gegenlesen der Arbeit danke ich Britta Pohlmann, Francis Marx und Angelika Deister.

Weiterhin danke ich den studentischen Hilfskräften Martin Luemkemann und Achim Luczyk, die mit der Anbindung der BRS-Datenbanken bzw. der Weiterentwicklung des Frontend-Applets einen wichtigen Beitrag zu dem Projekt „Rechercheassistent” geleistet haben.

Nicht zuletzt gilt mein Dank der Direktion und den Mitarbeitern der Universitätsbibliothek; in erster Linie Herrn Dr. Höppner, der als Projektpartner das Projekt mitinitiiert hat, aber auch Wolfgang Binder, Bernd Fehling, Friedrich Summann, Sabine Rahmsdorf und anderen, die bei konzeptionellen, organisatorischen, technischen und Layout-Fragen innerhalb des Projektes mitgewirkt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1

Einleitung

Im Informationszeitalter mit seinen ständig wachsenden Datenmengen ist spätestens seit dem Siegeszug des Internets nicht mehr das *Fehlen*, sondern das *Finden* von Informationen zum Problem geworden.

Bisher stellte die boolesche Logik die Grundlage sowohl für die Indexierung als auch für das spätere Wiederauffinden von Daten. So konnte in einer Volltextdatenbank nach Dokumenten gesucht werden, die z.B. die Stichworte „boolean", „information" *und* „retrieval" im Titel tragen, in einer Gebrauchtwagen-Datenbank nach Autos, die neuer als 5 Jahre sind, und in Datenbanken von Reisebüros nach Flugreisen nach Teneriffa, wobei das Ergebnis bestenfalls preislich sortiert ausgegeben werden konnte. Dokumente, bei denen nur zwei der drei Begriffe im Titel vorkommen, und der dritte nur im Untertitel, als „Schnäppchen" angebotene Autos, die nur einen Monat älter als 5 Jahre sind und Billig-Flüge auf das unmittelbar benachbarte La Gomera wurden von derartigen Systemen nicht gefunden.

Wenngleich die Unter- bzw. Fehl-Spezifikation der booleschen Suchanfragen als Ursache für das Scheitern einer derartigen Recherche herangezogen werden kann, so muss doch eine weitere Schlussfolgerung gezogen werden:

Die konventionelle, boolesche Suchlogik ist in vielen Fällen nicht mehr adäquat. Sie stößt an ihre Grenzen, da sie mit ihren beiden Verknüpfungen UND und ODER nur entweder zu spezielle oder zu allgemeine Anfragen unterstützt und nur die beiden extremen, in der Realität nicht genug differenzierenden Zustände „wahr" und „falsch" kennt, weswegen sie keine strukturierten Ergebnismengen ermöglicht.

Für moderne Suchmaschinen stellen sich drei neue Herausforderungen:

- Die Integration und Nutzbarmachung heterogener Datenquellen.
- Die Definition neuartiger, semantisch sinnvoller und intuitiv nachvollziehbarer Aggregationsoperatoren und Bewertungsfunktionen.
- Die Implementation effizienter Algorithmen, die trotz riesiger Datenvolumina und aufwändiger Suchlogik kurze Antwortzeiten garantieren.

Der erste Aspekt ist besonders im Bereich der Metasuche über verschiedene Internetquellen interessant. Mittlerweile steht eine Unzahl von Online-Datenbanken verschiedenster Domänen zur Verfügung. Viele davon basieren auf der konventionellen, booleschen Logik, weil die zugrundeliegenden, kommerziellen Datenbanken nur für dieses bestens erforschte Paradigma effiziente Datenstrukturen und Retrieval-Algorithmen zur Verfügung stellen. Der Anspruch für eine Metasuchmaschine besteht also nicht nur in der Homogenisierung der heterogenen Quellen, was zu einem großen Teil ein Problem eher technischer Natur ist, sondern insbesondere auch in der Abbildung neuartiger Retrieval-Techniken auf die althergebrachten, booleschen Systeme.

Der zweite Aspekt betrifft die mathematische Interpretation natürlichsprachlicher Anfragen wie der Suche nach einem „möglichst neuen und möglichst günstigen Gebrauchtwagen" oder „Dokumenten mit *einigen* der Begriffe ‚boolean', ‚information', ‚retrieval' ", die eine intuitivere und adäquatere Modellierung der Wirklichkeit liefern soll, als die, welche die boolesche Aussagenlogik zur Verfügung stellt. Sie muss mehreren Aspekten Rechnung tragen: Die Formulierung „möglichst neu" beinhaltet eine Toleranz, die in der booleschen Aussage „neuer als 5 Jahre" nicht gegeben ist. In der natürlichen Sprache ist inhärent ein Interpretations-Spielraum gegeben, der sich durch unterschiedliche *Zugehörigkeitsgrade* manifestiert: Ein Jahreswagen würde nach allgemeinem Verständnis wohl eindeutig das Kriterium „möglichst neu" erfüllen, wohingegen auf ein 3 Jahre altes Auto dieses Kriterium nur noch zu einem gewissen Grad zutrifft. Darüber hinaus impliziert die „Verknüpfung" ebendieser „weichen" Kriterien, aber auch bereits der boolesch interpretierbaren Kriterien wie „Dokument enthält Stichwort ‚retrieval' ", durch geeignete Quantoren eine eine ähnliche Toleranz: So sollen bei der obigen Anfrage an eine bibliographische Datenbank auch Dokumente geliefert werden, die nur zwei der drei gesuchten Stichworte enthalten. Der Datenbank soll darüber hinaus mitgeteilt werden können, welche der Stichworte besonders wichtig, und welche weniger wichtig sind. Die Toleranzen in allen diesen Aspekten natürlichsprachlicher Anfragen erfordern auch eine Toleranz, eine „Relevanz-Bewertung" der Ergebnismenge: Während bei der booleschen Logik Aussagen eindeutig „wahr" oder „falsch" sind, Datensätze also eindeutig zu der (unstrukturierten) Ergebnismenge gehörten oder eben nicht, ist nun eine strukturierte Ergebnisliste gewünscht, in der die Ergebnisse absteigend nach dem Grad ihrer Zugehörigkeit bzw. ihrer Relevanz oder auch ihrem „match" auf die Anfrage sortiert sind. Das Paradigma der Fuzzy-Logik liefert hierzu eine sinnvolle Erweiterung der booleschen Logik.

Der dritte Aspekt betrifft die triviale Forderung nach einer komfortablen Recherche mit kurzen System-Antwortzeiten, deren Erfüllung durch zweierlei Ansprüche erschwert wird: Die steigenden Datenvolumina ziehen unmittelbar einen höheren Aufwand bei der Indexierung

und Suche nach sich, den gestiegene Rechenleistung, Netz- und Speicherkapazität nur in begrenztem Umfang kompensieren können: Durch die Metasuche über verschiedene Internetquellen liegen die Daten nicht mehr wie früher lokal - und damit sehr schnell verfügbar - vor, und können aufgrund kürzerer Halbwertzeiten und Aktualisierungs-Intervalle auch nicht einfach gespiegelt werden. Zum anderen zieht der Ansatz der gewichteten Suche einen stark erhöhten Aufwand nach sich: Die Ergebnismengen bzw. zu aggregierenden Teilmengen enthalten nicht mehr, wie bei der booleschen Logik, genau die (wenigen) Datensätze, die das gewünschte Kriterium exakt erfüllen, sondern potentiell *alle* Datensätze der Datenbank, wobei für jeden einzelnen Datensatz der (häufig sehr geringe) Zugehörigkeitsgrad berechnet werden muss.

Die vorliegende Arbeit wird diese Problemstellungen näher beleuchten, bestehende Lösungsansätze vorstellen und eigene Ansätze darstellen. Sie grenzt sich ab von dem gut erforschten Bereich der auf boolescher Logik basierenden, konventionellen Datenbanken und von wissensbasierten bzw. solchen Systemen, die aufgrund ihres Ansatzes eine spezifische Wissensbasis (und sei es nur ein einfacher Thesaurus für ein bibliographisches Suchsystem) benötigen und damit sehr spezielle, dafür aber nur schwer übertragbare Lösungen für bestimmte, eng umrissene Domänen liefern. Vielmehr wird die allgemeine Fuzzy-Metasuche im Vordergrund stehen, die auf konventionellen Datenbanksystemen aufsetzt. Es sollen also keine Datenbank-internen Datenstrukturen untersucht werden, die eine effiziente, gewichtete Indexierung ermöglichen, aber eben Eingriff in die speziellen Datenbanken erfordern. Der Themenbereich von WWW-Metasuchmaschinen, die häufig nur die Ergebnisse anderer Suchmaschinen zusammenführen, selbst jedoch keine eigene Gewichtung bzw. Aggregation vornehmen, wird dabei gestreift, jedoch nicht den Schwerpunkt bilden. Als illustrierende Beispiele werden sich die bibliographische Suche und die Suche in einer Gebrauchtwagen-Datenbank, die bereits angerissen wurden, durch die Arbeit ziehen.

Die Arbeit ist entsprechend den eingangs dargestellten drei Hauptaspekten wie folgt aufgebaut: Kapitel 2 untersucht die generellen Problemstellungen der Fuzzy-Metasuche. Es findet eine grobe Aufwandsabschätzung statt, und es wird eine auf sog. Wrappern basierende 3-Schichten-Architektur vorgestellt, die den strukturellen Anforderungen an eine Metasuchmaschine gerecht wird. Das Streaming-Prinzip, auf welchem auch die späteren Aggregations-Algorithmen arbeiten, wird dargestellt, und grundlegende Nachbearbeitungsschritte, die der Heterogenität verschiedener Datenquellen entgegenwirken, untersucht.

Kapitel 3 legt mit einem kurzen Exkurs in die Fuzzy-Logik die Grundlagen für spätere Untersuchungen. Es werden zunächst intuitiv plausible und vollständige Kriterien für Quantoren und Bewertungsfunktionen für das bibliographische Retrieval aufgestellt, die dann formalisiert und diskutiert werden. Verschiedene Ansätze (t-, s-Normen, Generalized Quantifiers, FLQ, Σ-/FE-Count, OWA-Operatoren, p-Norm, WAO) werden auf diese Kriterien hin untersucht. Das Kapitel schließt mit der Herleitung eines Verfahrens zur formalen Optimierung von Anfragen, die aus o.g. Fuzzy-Operatoren aufgebaut sind.

In Kapitel 4 widmet die Arbeit sich in ihrem ausführlichsten Teil effizienten Algorithmen zur Aggregation von Fuzzy-Streams, die mittels der im vorigen Kapitel dargestellten Quantoren verknüpft werden sollen. Nach einer genaueren Problembeschreibung und der Analyse der wichtigsten Einflussfaktoren werden verschiedene Algorithmen zunächst theoretisch unter-

sucht und abschließend einem Praxistest in den dargestellten Szenarien der speziellen, bibliographischen Suche und der allgemeineren Fuzzy-Suche unterzogen.

Die Arbeit schließt mit einer Diskussion und einem Ausblick in Kapitel 5.

2

Generelle Problemstellungen der Fuzzy-Metasuche

2.1 Motivation und Kapitelübersicht

In diesem Kapitel werden generelle Problemstellungen, Anforderungen und Lösungsansätze der Informationssuche auf Basis von Fuzzy-Logik betrachtet, wobei ein besonderer Schwerpunkt auf Aspekten der Metasuche liegt. Neben Anforderungen an die Retrieval-*Qualität* - also z.B. die üblichen Qualitätskriterien „Precision" und „Recall" (vgl. z.B. [62]) - sind insbesondere zwei Aspekte für Fuzzy-Metasuchmaschinen von Bedeutung:

Beim gewichteten Retrieval müssen besondere Performance-Betrachtungen angestellt werden. Da Fuzzy-Mengen meistens sehr umfangreich sind, gilt es, effiziente Algorithmen zur Verarbeitung zu finden, und darüber hinaus, bereits durch die Systemarchitektur die technischen Voraussetzungen für die Skalierbarkeit des Systems zu schaffen. Dieser Aspekt wird in Unterkapitel 2.2 diskutiert.

Der zweite Aspekt, der bei Metasuchmaschinen relevant ist, betrifft die Integration heterogener Datenbanken in das System. Unterkapitel 2.3 wird näher auf die strukturellen Unterschiede und die unterschiedliche Mächtigkeit heterogener Datenbanken eingehen. Dabei werden vor allem Möglichkeiten und Grenzen von Nachbearbeitungsschritten untersucht.

2.2 Aufwandsabschätzung, Skalierbarkeit

Die beim gewichteten Retrieval auftretenden Ergebnismengen sind im Vergleich zu den Ergebnismengen des booleschen Retrievals in der Regel deutlich größer. Der Umfang einer Fuzzy-Menge entspricht dem der Disjunktion der booleschen Hüllen (vgl. Def. 3.3) aller Attribute, da beim gewichteten Retrieval auch solche Datensätze nachgewiesen werden sollen, bei denen einige Attribute nur zu einem geringen Grad zutreffen. Bei einer booleschen Anfrage, die in der Regel eine Baumstruktur aus Konjunktionen und Disjunktionen hat, wird der Umfang durch die Konjunktionen reduziert. Darüber hinaus liefern auch einzelne Fuzzy-Attribute bereits meist größere Ergebnismengen, wie das folgende Beispiel aus dem bibliographischen Retrieval verdeutlicht:

```
Stichwort 'fuzzy' UND 'retrieval' UND 'searchengine'
Datum     'moeglichst nach 1998'
```

Eine Interpretation dieser Anfrage auf Basis der Fuzzy-Logik soll auch Dokumente liefern, die kurz vor 1998 erschienen sind, und ebenso Dokumente, in denen z.B. nur zwei der drei Stichworte enthalten sind (mit entsprechend niedrigerer „Relevanz"). Vom Umfang der Ergebnismenge entspricht dies also - je nach Toleranz bzw. Restriktivität des Fuzzy-Operators „möglichst nach 1998" dem Umfang der folgenden booleschen Anfrage:

```
Stichwort 'fuzzy' ODER 'retrieval' ODER 'searchengine'
Datum     'nach 1995'
```

Unter Umständen vergrößert sich die Menge noch weiter, wenn die Operatoren noch weniger restriktiv formuliert sind - ein Operator „möglichst neu" soll evtl. auch noch Dokumenten von 1980 mit einer Relevanz größer 0.0 versehen.

Besonders kritisch wird dieses Problem bei sehr allgemeinen Stichworten in der Suche, die nicht zwangsläufig durch Stoppwortlisten eliminiert werden können. Tabelle 2.1 zeigt exemplarisch die 20 häufigsten Begriffe der mit gut einer Million Einträgen nur mittelgroßen Datenbank „Elsevier SCIENCE". Es ist offensichtlich, dass durch Verwendung solcher allgemeinen Begriffe in einer Fuzzy-Anfrage selbst bei kleinen bis mittelgroßen Datenbanken leicht Ergebnismengen von mehreren Hunderttausend Dokumenten entstehen können.

Eine zusätzliche Schwierigkeit gegenüber dem booleschen Retrieval liegt in der Anforderung, dass beim gewichteten Retrieval eine *strukturierte* Ergebnisliste geliefert werden soll, d.h. dass die Datensätze nach einer Relevanz sortiert präsentiert werden sollen.

Aus diesem Grund hat sich das das Prinzip des *Streamings* durchgesetzt. Streaming-Verfahren gehen davon aus, dass der Benutzer in der Regel zunächst nur an den ersten, hochrelevanten Datensätzen einer Ergebnismenge interessiert ist und ggf. weitere Ergebnisse iterativ anfordert. Anstatt also die Ergebnismenge als Ganzes zu übertragen, wird ein Stream-Objekt übergeben, von dem die Datensätze sequentiell angefordert werden können. Die interne Verarbeitung der Teilergebnisse basiert auf demselben Prinzip. Rechenzeit,

Speicherbedarf und vor allem die bei nicht lokal vorliegenden Datenbanken und verteilten Systemen nötige Übertragung per Netzwerk stellen dabei begrenzende Faktoren dar. Algorithmen zur Aggregation dieser Fuzzy-Streams werden in Kapitel 4 behandelt.

Rang	Stichwort	Anzahl an Dokumenten
1	study	207.665
2	high	196.557
3	effect	171.836
4	significant/-ce	155.831
5	analysis	149.098
6	mode	142.857
7	data	142.088
8	system	130.850
9	model	129.238
10	method	124.022
11	compare	122.077
12	structure	120.811
13	level	115.667
14	investigate	113.020
15	process	111.644
16	research	111.229
17	change	108.533
18	time	105.708
19	rate	103.836
20	result	100.714

Tabelle 2.1: Die 20 häufigsten Stichworte in der Datenbank „Elsevier SCIENCE". (Der Gesamtumfang der Datenbank beträgt gut 1 Mio. Dokumente). Der Aufwand bei der Berechnung komplexer Anfragen mit mehreren dieser allgemeinen Suchbegriffe kann leicht einige Hunderttausend Dokumente erreichen.

2.3 Heterogenität der Datenbanken

Bei der Heterogenität der Datenbanken, auf die eine Metasuchmaschine zugreift, können zwei Aspekte getrennt werden: Zum einen die durch unterschiedliche Architektur bedingten verschiedenen technischen Zugriffsmöglichkeiten und zum anderen die unterschiedliche Mächtigkeit, d.h. der „Funktionsumfang" bei der Formulierung einer Anfrage. Die Metasuchmaschine muss beide Aspekte berücksichtigen, um eine einheitliche Suche durchführen zu können.

2.3.1 Heterogene Architektur der Datenbanken

Die einer Metasuchmaschine zugrundeliegenden Datenbanken können über sehr unterschiedliche Architekturen verfügen. Dies äußert sich unter anderem in Syntax und Semantik der Anfragesprache, die von relationalen bzw. funktionalen Sprachen wie SQL[1] bzw. OQL[2] bis hin zu eher imperativen Sprachen wie BRS/Search[3] reichen. Auf die meisten Internet-Datenbanken kann ausschließlich über HTTP[4]/HTML[5]/CGI[6] zugegriffen werden [34].

Neben den verschiedenen Sprachen und Datenformaten ist ein grundlegender Unterschied, ob das verwendete Protokoll „zustandslos" (wie HTTP) oder „zustandsgebunden" (wie BRS/-Search) ist, d.h. ob Teilanfragen aufeinander aufbauen und die Ergebnisse iterativ abgefragt werden können, oder ob für jede Teilanfrage eine neue Datenbankverbindung geöffnet werden muss, die nach ihrer Abarbeitung nicht mehr zur Verfügung steht.

Des weiteren sollte berücksichtigt werden, dass gerade komplexere Anfragen von verschiedenen Datenbanken unterschiedlich performant bearbeitet werden können, so dass u.U. verschiedene Anfrageoptimierungen - abhängig von der Art der Datenbank - sinnvoll sind.

Zur Lösung dieser Probleme hat sich eine 3-Schichten- bzw. Mediatorarchitektur bewährt, bei der die Datenbanken durch sogenannte „Wrapper" an das System gekoppelt sind. Die Metasuchmaschine greift bei der Bearbeitung einer Benutzeranfrage nicht direkt auf die ausgewählten Datenbanken zu, sondern über diese „Umsetzer", die die spezifischen Details der Datenbanken kapseln und über eine einheitliche Schnittstelle zu der Metasuchmaschine verfügen (vgl. z.B. [11, 16]).

Abbildung 2.1 zeigt eine entsprechende Drei-Schichten-Architektur. Für jede Datenbank existiert ein Wrapper, der die Umsetzung der Teilanfragen in auf die jeweilige Datenbank angepasste Anfragen übernimmt. Weiterhin formt er die von der Datenbank gelieferten Ergebnisse in ein einheitliches Format um. Auf die Wrapper wird im weiteren Verlauf dieses Kapitels näher eingegangen.

[1] Structured Query Language[63]
[2] Object Query Language[54]
[3] Bibliographic Retrieval Service[7]
[4] Hypertext Transfer Protocol[37]
[5] Hypertext Markup Language[36]
[6] Common Gateway Interface[13]

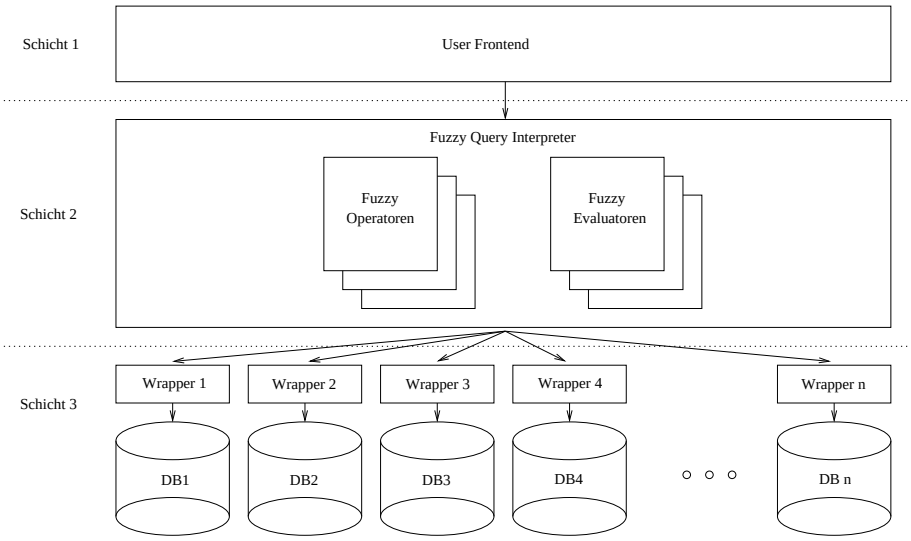


Abbildung 2.1: Drei-Schichten-Architektur einer Metasuchmaschine. Benutzerseitig steht eine über das WWW zugängliche Schnittstelle zur Verfügung. Die aufwendige Verarbeitung der Fuzzy-Mengen wird von der Hauptkomponente, dem „Fuzzy Query Interpreter" übernommen. Um diesem eine einheitliche Schnittstelle zu den heterogenen Datenbanken zur Verfügung zu stellen, dienen kompakte Wrapper.

Die Abarbeitung der Benutzeranfrage, das angesprochene Zerlegen in Teilanfragen und die Verarbeitung und Zusammenführung der Teilergebnisse in ein gewichtetes Gesamtergebnis übernimmt der „Fuzzy Query Interpreter". Er verfügt über Fuzzy-Operatoren („möglichst viele der Suchbegriffe x,y,z", „Erscheinungsdatum möglichst neu", etc.), die die Gewichtung einzelner Dokumente anhand ihrer Attribute definieren oder solche Gewichtungen aggregieren. Sie implementieren z.B. die üblichen min/max-Operatoren. Die Fuzzy-Evaluatoren, auf die der „Fuzzy Query Interpreter" zurückgreift, implementieren verschiedene Algorithmen bzw. Abarbeitungsstrategien, die - je nach Struktur der Anfrage - ausgewählt werden und eine effiziente Verarbeitung der Teilergebnismengen gewährleisten. Kapitel 3 wird die Fuzzy-Operatoren und Kapitel 4 die Fuzzy-Evaluatoren näher untersuchen.

Die obere Schicht stellt die Benutzerschnittstelle dar, über die per WWW auf das System zugegriffen werden kann.

Bei dem Design der Schnittstelle zwischen „Fuzzy Query Interpreter" und den „Wrappern" gilt es, Aufwand und Nutzen abzuwägen. Einerseits sollen die Wrapper möglichst klein und kompakt sein, damit neue Datenbanken ohne hohen implementatorischen Aufwand in das System integriert werden können, andererseits muss ein adäquater Funktionsumfang gewährleistet bleiben, der oft nur durch Nachbearbeitung sichergestellt werden kann. Auf solche Nachbearbeitungen, die in der Regel sehr rechenzeitintensiv sind, soll jedoch weitge-

hend verzichtet werden, weswegen die Schnittstelle zwischen „Fuzzy Query Interpreter” und Wrapper keine zu hohe Komplexität aufweisen darf.

Sich andererseits auf die „Schnittmenge” des Funktionsumfangs der Datenbanken zu beschränken, also nur solche Anfragen zu erlauben, die direkt von allen zugrundeliegenden Datenbanken unterstützt werden, bzw. sich 1:1 in von der Datenbank akzeptierte Anfragen übersetzen lassen, hat sich als zu restriktiv erwiesen.

„ADMIRE” [38] schlägt eine formale Beschreibung der Fähigkeiten heterogener Datenbanken vor, um Anfragen von einem universellen Mediatorformat in das spezifische Datenbankformat umwandeln zu können. Aber auch dieses System kann nicht vollständig auf Nachbearbeitungen verzichten und scheitert trotz dieser Nachbearbeitungen - die sich in einigen Fällen als zu rechenzeitintensiv erweisen haben - an bestimmten Kombinationen von Anfragen.

Bei der bibliographischen Suchmaschine „Rechercheassistent” wird eine hybride Lösung verwendet. Als sinnvoller Mittelweg hat es sich bewährt, eine generelle boolesche Anfragesyntax vorauszusetzen, wobei die maximale Komplexität des Syntaxbaumes, die der Wrapper verarbeiten kann, je nach Datenbank variieren darf (Notation in EBNF):

```
<request>      = <query> <sortcriteria>
<query>        = <conjunction> | <disjunction> | <negation> | <atomicquery>
<conjunction>  = "(" <query> { "AND" <query> } ")"
<disjunction>  = "(" <query> { "OR" <query> } ")"
<negation>     = "NOT (" <query> ")"
<atomicquery>  = <titlekeyword> | <author>
<sortcriteria> = <attribute>
```

Für das Zusammensetzen einer Teilanfrage stellt jeder Wrapper dem „Fuzzy Query Interpreter” eine Abfragemöglichkeit zur Verfügung, wann die kritische Komplexität, die von der jeweiligen Datenbank nicht mehr verarbeitet werden kann, überschritten wird. Die Anfrage wird notfalls auf eine Konjunktion von positiven Literalen reduziert, welches somit die minimale Anforderung an die vom Wrapper zu verarbeitende Syntax darstellt:

```
<query>        = <conjunction>
<conjunction>  = <atomicquery> { "AND" <atomicquery> }
<atomicquery>  = <titlekeyword> | <author>
```

Zusätzlich dazu kann die „Toleranz”, die ein Wrapper beim matching von Stichworten erlauben soll, in Form eines skalaren Faktors übergeben werden. Durch welche Methoden bzw. linguistischen Verfahren (Suffixlisten, approximatives Matching, Adjazenzoperatoren, Komposita-Zerlegung, Wortstammbildung, morphologische Analyse, Thesauri, Expertenwissen, n-Gramme, vgl. z.B. [62, 31, 41, 35, 55]) die Datenbank dies realisiert, wird dabei nicht näher spezifiziert, da die Methoden in ihren Variationen recht zahlreich sind und eine spezifischere und trotzdem einheitliche Behandlung über die Datenbanken hinweg dadurch

ausgeschlossen ist. Der Grad der Toleranz kann dabei jedoch beliebig genau abgestuft sein - eine Datenbank, die verschiedene Verfahren unterstützt, kann mit steigendem Toleranzfaktor mehrere der Verfahren geeignet kombinieren.

Als weitere Anforderung benötigen verschiedene Algorithmen des „Fuzzy Query Interpreters" eine Sortierung des Ergebnisses bzw. eine Suche darauf anhand von skalaren Attributen wie dem Erscheinungsdatum. Dies ist bei einigen Datenbanken nicht vollständig ohne Nachbearbeitung möglich, deren Aufwand jedoch relativ gering ist, wie das folgende Kapitel zeigen wird.

2.3.2 Unterschiedliche Mächtigkeit, Nachbearbeitungsschritte

Neben den unterschiedlich mächtigen Anfragesprachen, auf die im vorherigen Kapitel eingegangen wurde, verfügen verschiedene Datenbanken über unterschiedliche Mächtigkeit was den Funktionsumfang betrifft. Um von einer Datenbank nicht direkt unterstützte Eigenschaften nachträglich aufzusetzen, werden hier entsprechende Verfahren zur Nachbearbeitung diskutiert. Der Wrapper einer Datenbank, die eine Anfrage nicht direkt verarbeiten kann, stellt also eine reformulierte Anfrage und bearbeitet das von der Datenbank gelieferte Ergebnis so nach, dass das Endergebnis der ursprünglichen Anfrage genügt.

Ein konkretes Problem, welches sich bei Datenbanken häufig stellt, ist eine unvollständige Sortierung. Zum einen ist dies durch fehlende Attribut-Einträge zu erklären, zum anderen dadurch, dass eine Datenbank nach einem anderen Attribut sortiert ist, welches aber mit dem gewünschten Attribut korreliert.

Im folgenden wird dies anhand des Beispiels des Erscheinungsdatums eines Dokumentes erläutert. Dieses korreliert häufig mit dem Datum, an dem das Dokument in die Datenbank aufgenommen wurde, welches zwar nicht direkt verfügbar ist, aber seinerseits üblicherweise mit der Dokument-ID korreliert, nach der die Dokumente in der Datenbank sortiert abgelegt sind. Mit anderen Worten, alte Dokumente haben normalerweise eine kleine ID, neuere Dokumente eine größere. Durch Löschungen von Einträgen und „Auffüllen" der dadurch entstehenden Lücken durch neue Dokumente oder durch Überschreiben bei Erscheinung einer neuen Auflage entstehen „Ausreißer nach oben", also Dokumente mit niedriger ID, aber aktuellem Datum. Durch späteres Aufnehmen eines Dokumentes in die Datenbank kommt es zu „Ausreißern nach unten", also Dokumenten mit hoher ID, aber älterem Datum. Tabelle 2.2 gibt Aufschluss über die Häufigkeit fehlerhafter Datumseinträge einiger Datenbanken.

Die beiden folgenden Kapitel diskutieren, wie der Aufwand, der für das Suchen und das Nachsortieren auf diesen „semi-sortierten", d.h. teilweise sortierten Datenbanken erforderlich ist, durch entsprechende Algorithmen auf ein Minimum gedrosselt werden kann, und wie die für diese Verfahren nötigen Parameter anhand von Heuristiken gefunden werden können.

Ein weiteres, zentrales Problem ist die Implementation von Negationen und Disjunktionen, von denen insbesondere erstere für die Realisierung einer Fuzzy-Metasuchmaschine unumgänglich sind. Werden Negationen von einer Datenbank nicht unterstützt, so müssen

	Fehlender Eintrag	Ausreißer nach oben	Ausreißer nach unten	Fehler gesamt
Elsevier SCIENCE	2.93%	0.00%	0.00%	2.93%
JADE	1.19%	0.00%	0.00%	1.19%
JASON	1.79%	28.46%	28.95%	59.20%
OPAC	13.71%	0.16%	0.02%	13.90%
Springer	4.66%	0.00%	0.00%	4.66%

Tabelle 2.2: Im Vergleich zur Dokument-ID fehlerhafte Datumseinträge in verschiedenen Datenbanken. Bis zu knapp 60% der Dokumente sind ihrem Datum entsprechend nicht korrekt einsortiert und müssen in einem Nachbearbeitungsschritt gesondert sortiert werden.

sie nachträglich durch den zugehörigen Wrapper durchgeführt werden. Unterkapitel 2.3.2.4 wird das Laufzeitverhalten dieser Nachbearbeitungen untersuchen.

2.3.2.1 Suche auf semi-sortierten Datenbanken

Anfragen, die nicht direkt von jeder Datenbank verarbeitet werden können, sind meist solche, die ein Feld auf ein Intervall einschränken, für welches keine indizierte Datei vorliegt, nach dem die Datenbank also nicht sortiert ist. Oben genannte Datenbanken sind beispielsweise nicht nach Datum sortiert und ermöglichen daher auch keine direkte Suche darauf, wie etwa

```
"Stichwort 'fuzzy' UND Datum nach 1990"
```

Stattdessen muss von dem Wrapper auf dem gelieferten Ergebnis der direkt ausführbaren Teilanfrage (`"Stichwort 'fuzzy'"`), welches nach der ID (die mit dem Erscheinungsdatum korreliert) sortiert ist, nachträglich das Dokument gesucht werden, ab dem das Kriterium (`"Datum nach 1990"`) erfüllt ist und somit das Ergebnis der Gesamtanfrage vorliegt. Dies setzt natürlich voraus, dass die Datenbank einen „random access" (wahlfreien Zugriff) auf die Ergebnisliste ermöglicht. Eine solche Suche setzt also bereits auf lowlevel-Ebene *innerhalb* des Wrappers - also noch *vor* Umwandlung des Ergebnisses in einen sequentiell abgreifbaren Stream - an.

Als Suchalgorithmus auf einer sortierten Liste ist die binäre Suche mit einem mittleren Aufwand von $O(log_2(n))$ optimal. Auf einer „semi-sortierten" Liste - wie sie im o.g. Szenario vorliegt - kann bei der Binärsuche selbst ein einziger Ausreißer im schlimmsten Fall dazu führen, dass gar kein Ergebnis gefunden wird. Das folgende Beispiel verdeutlicht dies:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Wert	1	1	1	6	2	2	3	3	3	4	4	5	5	6	6	7

Wird in dieser Liste das erste Vorkommen der Zahl 2 gesucht, so beginnt die Binärsuche auf der Hälfte, also an Position 8. Da der gesuchte Wert 2 kleiner ist als der Wert an dieser Position, nämlich 3, wird das Intervall halbiert, und an Position 4 weitergesucht. Dort befindet sich der Ausreißerwert 6, der den Algorithmus fälschlicherweise links dieser Position weitersuchen lässt. Als Folge davon wird kein Ergebnis gefunden.

Eine lineare Suche würde mit Sicherheit zum Erfolg führen, jedoch beträgt ihr Aufwand O(n), was aus eingangs genannten Gründen nicht akzeptabel ist.

Eine Modifikation der Binärsuche genügt jedoch, um eine erfolgreiche Suche mit akzeptablem Laufzeitverhalten durchführen zu können: Anstelle eines einzelnen Wertes an der jeweiligen Position wird ein „Fenster" von mehreren Werten betrachtet. Von diesen Werten muss ein Mittelwert gebildet werden, der ein sinnvolles Ergebnis liefert. Anstelle des gewöhnlichen arithmetischen Mittels, bei dem einzelne, sehr stark abweichende Ausreißer das Ergebnis zu stark verfälschen, bietet sich der sog. Median an, der den in der Rangfolge in der Mitte stehenden Wert, also in diesem Fall an Position 4 bei zusätzlicher Berücksichtigung der beiden direkt benachbarten Zahlen 1 und 2 korrekterweise 2 wählt. (In der Tat würde bei das arithmetische Mittel bei $(1 + 6 + 2)/3 = 3$, also zu hoch liegen.)

Eine geeignete Fenstergröße kann statistisch abgeschätzt werden: Sei p die Wahrscheinlichkeit für einen Ausreißer, wobei hier der *worst case* betrachtet wird, bei dem *alle* Ausreißer in eine Richtung weisen (innerhalb eines Fensters gleichen sich Ausreißer nach oben und Ausreißer nach unten gegenseitig aus und verringern damit die Fehlerwahrscheinlichkeit). Weiterhin sei w die Fenstergröße und P die Fehlerwahrscheinlichkeit, dass der Algorithmus scheitert.

Geht man davon aus, dass sich von den korrekten Werten *innerhalb* des Fensters nur wenige unterscheiden, so wird der Median offensichtlich genau dann nach oben bzw. unten verfälscht, wenn mehr als die Hälfte der Werte im Fenster Ausreißer sind. (Die Annahme, dass nur wenige unterschiedliche, korrekte Werte in dem Fenster liegen, ist in der Praxis für Fenstergrößen, die klein gegenüber der gesamten Datenbankgröße sind, sicher richtig, da z.B. bei einer Datenbank mit einer Million Dokumenten bei 100 aufeinanderfolgenden Dokumenten nur wenige aus unterschiedlichen Jahren - vermutlich nicht einmal aus unterschiedlichen Monaten - stammen.)

Daher berechnet sich P nach Bernoulli wie folgt:

$$P(p, w) = \sum_{i=\frac{w}{2}+1}^{w} \binom{w}{i} p^i (1-p)^{w-i}$$

Wie in Abbildung 2.2 zu erkennen, ist die Fehlerwahrscheinlichkeit P erwartungsgemäß bei hohen Ausreißerwahrscheinlichkeiten p und kleinen Fensterbreiten w mit einem recht hohen Wert von etwa 8 Prozent am größten. Abbildung 2.3 zeigt jedoch deutlich, dass sie sich - auch bei der größten vorkommenden Wahrscheinlichkeit ($p = 0.3$ für Ausreißer in eine

Richtung in der Datenbank JASON) durch genügend große Fensterbreiten ($w = 31$) leicht auf sehr kleine Werte von unter 1% oder mit $w = 55$ auf unter 0.1% drücken lässt. Bei kleineren Ausreißerwahrscheinlichkeiten (z.B. $p = 0.14$ für die Datenbank OPAC) ist die Fehlerwahrscheinlichkeit auch bei kleinen Fensterbreiten von $w = 7$ mit unter einem Prozent bzw. unter 0.1% für $w = 13$ unkritisch.

Das optimale Laufzeitverhalten der Binärsuche, $O(log_2(n))$ kann also auf Kosten einer sehr geringen Fehlerwahrscheinlichkeit aufrecht erhalten werden.

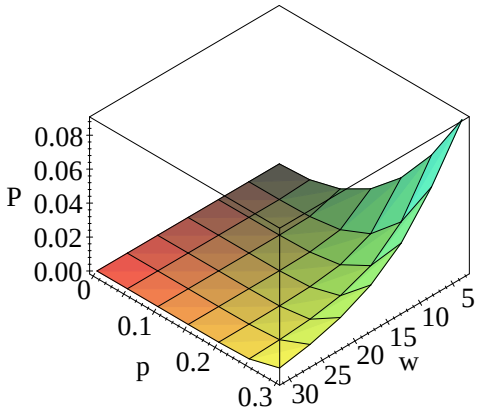


Abbildung 2.2: Fehlerwahrscheinlichkeit der Binärsuche mittels Median. Auch bei relativ hohen Ausreißerwahrscheinlichkeiten kann die Fehlerwahrscheinlichkeit durch genügend große Fensterbreiten auf akzeptable Werte reduziert werden.

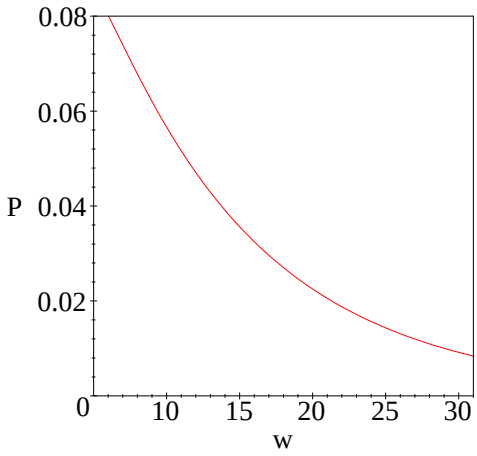


Abbildung 2.3: Fehlerwahrscheinlichkeit der Binärsuche mit Median für $p = 0.3$. Selbst für diese hohe Ausreißerwahrscheinlichkeit kann die Fehlerwahrscheinlichkeit durch eine Fensterbreite von $w = 31$ auf unter 1% gebracht werden.

2.3.2.2 Sortierung einer semi-sortierten Datenbank

Muss ein Ergebnis-Stream nach einem Attribut sortiert ausgegeben werden, nach welchem die Datenbank nicht direkt indexiert ist, so kann dies bei einer semi-sortierten Datenbank durch eine Nachbearbeitung erreicht werden:

Generell hat das Sortieren einer Menge von Datensätzen eine Komplexität von $O(n*log_2(n))$. Da wie eingangs erläutert die Ergebnismengen beim gewichteten Retrieval oft aus mehreren 100.000 Datensätzen bestehen, muss auf allgemeine Sortierverfahren jedoch verzichtet werden. Des weiteren widerspricht eine konventionelle Sortierung dem Streaming-Prinzip, da für den kompletten Sortiervorgang bereits alle Datensätze vorliegen müssen, bevor das erste Ergebnis ausgegeben werden kann. Gewünscht ist ein Algorithmus, bei dem bereits nach Einlesen nur eines kleinen Teils des Eingangs-Streams Ergebnisse auf den Ausgangs-Stream geleitet werden können.

Da in diesem Kapitel von „semi-sortierten" Streams ausgegangen wird, ist keine komplett neue Sortierung mehr nötig, vielmehr müssen die Ergebnisse nur „nachsortiert" werden. Dazu bietet es sich an, einen Puffer variabler Größe anzulegen, in den die eingelesenen Datensätze sortiert abgelegt werden. Wird die Größe des Puffers überschritten, so wird der erste Datensatz aus dem Puffer entfernt und in den Ausgabe-Stream geleitet. Das folgende Beispiel verdeutlicht dies. Wie zu erkennen, befindet sich an Position 2 des Streams ein Ausreißer nach oben, an den Positionen 6 und 14 jeweils ein Ausreißer nach unten, und

Position 8 und 9 sind vertauscht:

Eingangs-Stream:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert	1	6	2	2	3	1	3	4	3	4	5	5	6	1	6

In diesem Beispiel wird ein konstanter Puffer für 5 Datensätze verwendet. Nach Einlesen der ersten 5 Werte wird noch kein Ergebnis ausgegeben, lediglich die ersten Werte sind in den Puffer einsortiert worden:

Eingangs-Stream:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert	-	-	-	-	-	1	3	4	3	4	5	5	6	1	6

Puffer:

Position	1	2	3	4	5
Wert	1	2	2	3	6

Zum Einlesen des nächsten Wertes (1) vom Eingangs-Stream muss Platz im Puffer geschaffen werden, und der erste Wert im Puffer (1) wird auf den Ausgabe-Stream gegeben. Danach wird der Wert 3 gelesen, und die zuvor eingelesene 1 aus dem Puffer ausgegeben. Nach dem 5. Zwischenschritt stellt sich der Zustand wie folgt dar:

Eingangs-Stream:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert	-	-	-	-	-	-	-	-	-	-	5	5	6	1	6

Puffer:

Position	1	2	3	4	5
Wert	3	3	4	4	6

Ausgangs-Stream:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert	1	1	2	2	3	-	-	-	-	-	-	-	-	-	-

Wenn der Eingangs-Stream vollständig abgearbeitet ist und auch der Puffer komplett auf den Ausgabe-Stream gelenkt wurde, ist das Ergebnis das folgende:

Ausgangs-Stream:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert	1	1	2	2	3	3	3	4	1	4	5	5	6	6	6

Wie zu erkennen, ist das Ergebnis *nicht vollständig* sortiert. Der Ausreißer, der sich vor der Sortierung an Stelle 14 befand, ist nun auf Position 9, jedoch nicht korrekt an eine der ersten Stellen gerückt. Davon abgesehen ist die Liste vollständig sortiert.

Leider ist es nicht möglich sicherzustellen, dass alle Ausreißer korrekt einsortiert werden: Im worst case befindet sich der Datensatz, welcher korrekterweise als erstes ausgegeben werden müsste, als Ausreißer an letzter Stelle im Eingangs-Stream. Um ihn mit Sicherheit korrekt einsortieren zu können, müsste also von vorneherein der komplette Stream eingelesen werden, was gerade vermieden werden muss.

Es liegt also in der Natur der Problemstellung, dass Ausreißer nach unten - also niedrige Werte, die an hoher Position kommen, generell kaum zu handhaben sind. Dahingegen können Ausreißer nach oben, also hohe Werte, die an zu niedriger Position stehen, sehr gut behandelt werden, sie verbleiben entsprechend lange im Puffer (eine adäquate Puffergröße vorausgesetzt). Bei einer absteigenden Sortierung, wie z.B. bei der natürlichen Sortierung der Ergebnisse nach Relevanz verhält es sich umgekehrt, aber ansonsten analog.

Eine noch offene Fragestellung ist, wie fehlende Werte (um bei dem eingangs angeführten Beispiel zu bleiben, also Dokumente, für die kein Datum verfügbar ist) behandelt werden sollen. Wird ihnen generell eine hohe Relevanz zugewiesen, so sind es Ausreißer nach oben, und sie werden in der Regel zu spät ausgegeben. Wird ihnen eine niedrige Relevanz zugewiesen - was auch eher der natürlichen Intuition entspricht - so werden sie korrekt einsortiert.

Zu lösen bleibt eine geeignete Wahl der Puffergröße. Offensichtlich ist es nicht sinnvoll, wie im Beispiel eine konstante Größe zu verwenden, da der Umfang eines Ergebnis-Streams nicht zwangsläufig im Voraus bekannt ist. Ein Puffer, der mit fortschreitender Abarbeitung des Streams proportional mitwächst, ist hier angemessen. Allerdings sollte die Puffergröße einen bestimmten Minimalwert nicht unterschreiten - es ist also ein Initialwert festzulegen. Ein sinnvoller Proportionalitätsfaktor und initiale Puffergröße werden im folgenden statistisch bestimmt.

Sei b die initiale Puffergröße, f der Proportionalitätsfaktor, unter dem der Puffer mit fortschreitender Abarbeitung des Streams mitwächst, n die Anzahl bereits eingelesener Datensätze und p die Wahrscheinlichkeit für einen zu puffernden Ausreißer. Damit berechnet sich die aktuelle Puffergröße als Maximum der Initialgröße und der „proportionalen Größe" als $max(b, f * n)$. Dann ist P, die Fehlerwahrscheinlichkeit, dass der Puffer überläuft, also gleich der Wahrscheinlichkeit, dass mehr Ausreißer unter den bereits eingelesenen Datensätzen vorhanden sind, als der Puffer fassen kann:

$$P(b, f, n, p) = \sum_{i=max(b, f*n)+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

Abbildung 2.4 zeigt die Fehlerwahrscheinlichkeit P in Abhängigkeit der initialen Puffergröße b und des Proportionalitätsfaktors f für $p = 0.3$ und $n = 5$, $n = 50$ und $n = 500$.

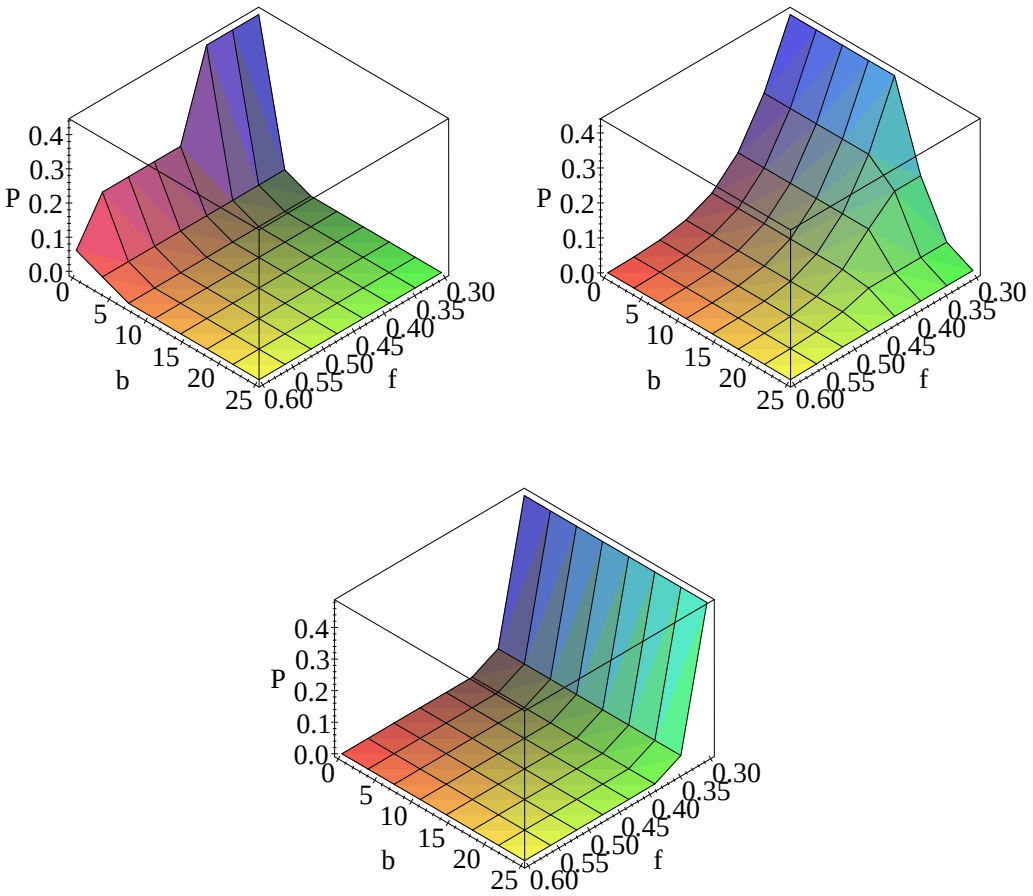


Abbildung 2.4: Fehlerwahrscheinlichkeit von „Buffer-Sort" bei $p = 0.3$ und $n = 5$ (oben links), $n = 50$ (oben rechts) bzw. $n = 500$ (unten) in Abhängigkeit des Proportionalitätsfaktors f und der initialen Puffergröße b.

Wie aus den Diagrammen zu erkennen ist, bestätigt sich die Erwartung, dass eine große initiale Puffergröße b *alleine* nicht ausreicht, um für große Werte von n die Fehlerwahrscheinlichkeit P auf ein akzeptables Maß zu reduzieren: Abbildung 2.4(c) verdeutlicht, dass bei großen n kaum mehr eine Abhängigkeit zwischen initialer Puffergröße b und Fehlerwahrscheinlichkeit P besteht.

Analog reicht bei kleinen n (also zu Anfang des Streams, vgl. Abbildung 2.4(a)) ein hoher Proportionalitätsfaktor f *alleine* (also $b = 0$) nicht aus, um die Fehlerwahrscheinlichkeit P auf deutlich unter 10% zu reduzieren.

Die *Kombination* adäquater Werte liefert jedoch sinnvolle Ergebnisse. Wie aus den obigen Betrachtungen ersichtlich, liegt die maximale Fehlerwahrscheinlichkeit P bezüglich n auf der Grenze zwischen initialer Puffergröße b und dynamischer Puffergröße $f * n$, also bei $b = f * n \Leftrightarrow n = \frac{b}{f}$:

$$\begin{aligned} P_{n_{max}}(b, f, p) &= \sum_{i=max(b, f*\frac{b}{f})+1}^{\frac{b}{f}} \binom{\frac{b}{f}}{i} p^i (1-p)^{\frac{b}{f}-i} \\ &= \sum_{i=b+1}^{\frac{b}{f}} \binom{\frac{b}{f}}{i} p^i (1-p)^{\frac{b}{f}-i} \end{aligned}$$

Abbildung 2.5 zeigt die maximale Fehlerwahrscheinlichkeit $P_{n_{max}}$ in Abhängigkeit von initialer Puffergröße b und Proportionalitätsfaktor f, welcher jeweils auf das Intervall $[p..2p]$ beschränkt wurde. Beide Diagramme weichen nur äußerst geringfügig voneinander ab, d.h. für nahezu *beliebige* Ausreißerwahrscheinlichkeiten p kann b z.B. auf 35 (bzw. 40) und f auf $\frac{3}{2}p$ (bzw. $\frac{5}{3}p$) festgelegt werden, was die Fehlerwahrscheinlichkeit P auf ein akzeptables Maß von unter 1% (bzw. 0.1%) reduziert.

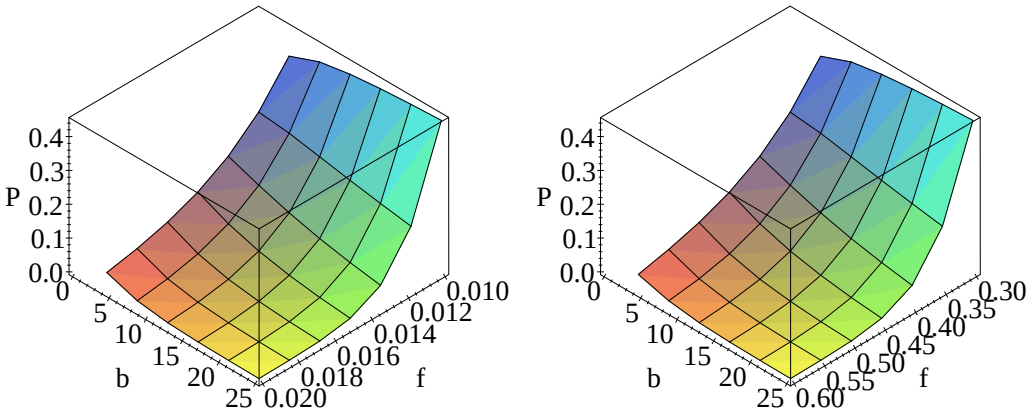


Abbildung 2.5: Fehlerwahrscheinlichkeit von „Buffer-Sort" projiziert auf $n_{max} = \frac{b}{f}$) für $p = 0.01$ bzw. 0.3. Durch eine geeignete Kombination von initialer Puffergröße b und Proportionalitätsfaktor f kann die Fehlerwahrscheinlichkeit auf ein unkritisches Maß reduziert werden.

2.3.2.3 Nachbildung von „random accesses"

Neben den bisher geschilderten Verfahren, die auf dem Streaming-Prinzip basieren, also einen „sorted access" realisieren, benötigen einige der in Kapitel 4 vorgestellten Algorithmen einen „random access", einen wahlfreien Zugriff, mit dem überprüft wird, ob ein Datensatz mit einer gegebenen ID zu einer (Fuzzy-)Ergebnismenge gehört, bzw. welchen Zugehörigkeitsgrad er zu dieser Menge hat.

Ein random access auf einer Fuzzy-Menge, die durch eine Aggregation n anderer (Fuzzy-) Mengen gebildet wurde, kann dabei direkt zurückgeführt werden auf n random accesses auf die zugrundeliegenden Fuzzy-Mengen. Nachgebildet werden müssen also die atomaren, datenbankseitigen random accesses. Dafür gibt es im Prinzip zwei Möglichkeiten, den „direct access" und den „echten" random access.

Direct access

Sofern es die Ressourcen erlauben und die Datensätze eine gewisse Größe nicht überschreiten, können anstatt der IDs der Datensätze auch die kompletten Datensätze inklusive aller benötigten Attribute direkt im Speicher gehalten und durch die Streams übergeben werden. Mittels eines „direct access" kann dann direkt auf den Zugehörigkeitsgrad eines Datensatzes zu einer (Fuzzy-)Menge zugegriffen werden, ohne dass zusätzliche Laufzeitkosten in Form von Datenbankzugriffen entstehen.

Beim bibliographischen Retrieval ist beispielsweise das Datumsfeld ein derartiges Attribut. Da es i.d.R bei einer tabellarischen Anzeige der Ergebnismenge ohnehin ausgegeben wird, und damit von der Datenbank angefordert werden muss, und da der Übertragungs- und Speicheraufwand minimal ist, ist es sinnvoll, dieses Attribut von vorneherein von der Datenbank anzufordern und im Speicher zu behalten. Der Zugehörigkeitsgrad eines Dokumentes zu der Fuzzy-Menge der „möglichst neuen" Dokumente kann dann ohne einen zusätzlichen Datenbankzugriff bestimmt werden.

Dies ist jedoch nicht bei allen Attributen möglich oder sinnvoll, z.B. können die Attribute aus verschiedenen Quellen stammen. Die prinzipielle Möglichkeit für einen derartigen direkten Zugriff kann jedoch bereits bei Verwendung nur einer Datenbank eingeschränkt sein, wenn das Attribut lediglich über einen inversen Index zugänglich ist, und bei der Ausgabe nicht mit zurückgegeben wird.

Unter Umständen kann z.B. die einfache Stichwortsuche nicht sinnvoll durch einen direct access realisiert werden: Sind in einer Datenbank z.B. nur die üblichen Felder „Titel", „Autor", „ Verlag", „Erscheinungsjahr" und dergleichen verfügbar, und existiert zusätzlich ein inverser Index über vorkommende Stichworte im Volltext oder Abstract, so ist der direkte Zugriff auf letztere Attribute überhaupt nicht möglich.

Selbst in dem Fall, dass der Volltext bzw. das Abstract komplett in der Datenbank abgelegt sind, kann ein direct access aus zwei Gründen problematisch sein.

Zum einen muss der Matching-Mechanismus der Datenbank komplett reimplementiert werden. Bei der einfachen, booleschen Stichwortsuche, die möglicherweise noch Trunkierung verwendet, ist dies noch unproblematisch. Greift die Datenbank jedoch auf weitergehende Methoden wie fehlertolerantes matching, Adjazenzoperatoren oder gar Thesauri zurück, so kann der implementatorische Aufwand enorm sein. Darüber hinaus kann in diesem Fall nur schwer sichergestellt werden, dass der direct access auch tatsächlich exakt mit dem Matching-Mechanismus der Datenbank übereinstimmt, was für konsistente Ergebnisse zwingend notwendig ist.

Das zweite Problem betrifft neben dem Speicheraufwand vor allem die Übertragung über das Netzwerk, welches in vielen Fällen einen Engpass darstellt. Wie geschildert muss für einen direct access anstatt der einfachen ID, die üblicherweise nicht mehr als 4 oder 8 Bytes umfasst, der komplette Datensatz angefordert und im Speicher gehalten werden. Die mittlere Größe derartiger Datensätze lässt sich nur schwer einschätzen und schwankt je nach Anwendung erheblich. Beim bibliographischen retrieval mit Dokumenten, die neben den o.g. Attributen z.B. noch das Abstract enthalten, sind Größenordnungen von einigen Kilobytes nicht selten. Es entsteht also ein Overhead vom Faktor 1000 oder mehr. Bei Multimedia-Anwendungen mit Bild-, Audio- oder gar Video-Dateien ist dieser Overhead noch um Größenordnungen höher. Abgesehen davon setzt der kalkulatorische Aufwand hier praktische Grenzen.

Random access

Ein echter „random access" kann von einer Datenbank auf unterschiedliche Weise unterstützt werden. Wird er unmittelbar unterstützt, so kann er direkt an die Datenbank delegiert

werden. Es würde dann eine Anfrage der Form „(Zu welchem Grad) gehört Datensatz <*id*> zu Menge <M>?" gestellt werden.

Oft wird ein random access jedoch nur mittelbar unterstützt, d.h. es wird von der Datenbank zu einer Anfrage eine Referenz auf eine nach IDs sortierte Ergebnismenge geliefert, auf die dann wahlfrei zugegriffen werden kann. In diesem Fall ist eine binäre Suche nach dem gewünschten Datensatz nötig. Diese bringt einen logarithmischen Aufwand mit der Größe der Ergebnismenge mit sich, und ist daher nur bei kleinen Datenbanken empfehlenswert.

2.3.2.4 Nachbildung boolescher Negationen und Disjunktionen

Die Nachbildung von booleschen Negationen (genauer: von konjunktiver Verknüpfung einer Anfrage mit einer zusätzlichen Negation) und Disjunktionen kann relativ unproblematisch implementiert werden, wie in diesem Kapitel gezeigt wird.

Um eine Anfrage der Form $A \wedge \neg B$ zu bearbeiten, müssen beide Teilanfragen einzeln ausgeführt und im Nachbearbeitungsschritt ihre Differenz gebildet werden, also lediglich die Datensätze aus A ausgegeben werden, die *nicht* in B vorkommen. Um den Overhead möglichst gering zu halten, sollte als zu subtrahierende Teilanfrage jedoch nicht alleine B ausgeführt werden, sondern $A \wedge B$. (Aufgrund des Kontradiktions-Gesetzes ist das Resultat von $A \wedge \neg(A \wedge B)$ dasselbe, wie von $A \wedge \neg B$.) Beide Ergebnis-Streams können dann parallel, iterativ abgearbeitet werden. Voraussetzung ist lediglich eine einheitliche Sortierung.

Als Maß für den nötigen Aufwand dient der Quotient q, der die Anzahl an zu betrachtender Datensätze (als Summe der von beiden Streams angeforderten Datensätze) für einen auszugebenden Datensatz (als Differenz der von beiden Streams angeforderten Datensätze) ausdrückt. Mit N als Anzahl der Datensätze in der Datenbank berechnet sich dieser Quotient q wie folgt:

$$
\begin{aligned}
q &= \frac{\#\textit{betrachtete Datensätze}}{\#\textit{auszugebende Datensätze}} \\
&= \frac{\#\textit{Datensätze aus } A + \#\textit{Datensätze aus } A \cap B}{\#\textit{Datensätze aus } A - \#\textit{Datensätze aus } A \cap B} \\
&= \frac{|A| + |A \cap B|}{|A| - |A \cap B|} \\
&= \frac{N * P(A) + N * P(A \wedge B)}{N * P(A) - N * P(A \wedge B)} \\
&= \frac{P(A) + P(B|A) * P(A)}{P(A) - P(B|A) * P(A)} \\
&= \frac{1 + P(B|A)}{1 - P(B|A)}
\end{aligned}
$$

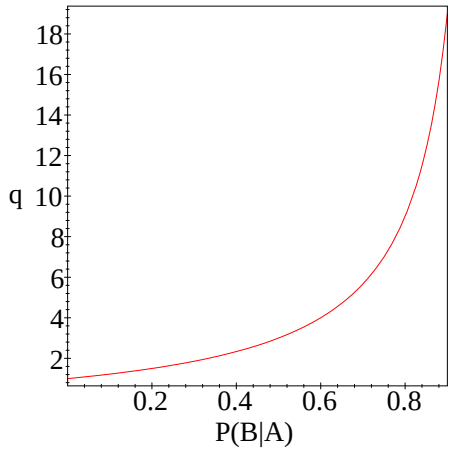


Abbildung 2.6: Aufwand für die Nachbearbeitung einer booleschen Negation. Gezeigt ist die Anzahl zu betrachtender Dokumente für ein auszugebendes Dokument bei einer Anfrage der Form $A \wedge \neg B$ in Abhängigkeit der relativen Wahrscheinlichkeit $P(B|A)$. Für in der Praxis vorkommende $P(B|A)$ bis zu 0.5 liegt der Wert unter 3 im praktikablen Intervall.

Dieser Quotient ist in Abbildung 2.6 graphisch dargestellt. Leider ist die relative Wahrscheinlichkeit $P(B|A)$, also die Wahrscheinlichkeit für das Vorkommen eines Suchbegriffes B unter der Bedingung des Auftretens des Begriffes A nur schwer einzuschätzen, und insbesondere stark von der Generalität der Stichworte - also letztendlich von der Suchstrategie des Benutzers - abhängig. Geht man jedoch von plausiblen Werten für $P(B|A)$ von unter 0.5 aus, so ist ersichtlich, dass für ein ausgegebenes Dokument maximal 3 Dokumente angefordert werden müssen, was für konventionelle Datenbanken ein absolut unproblematischer Faktor ist. Auch für extrem hohe Werte für $P(B|A)$ wie 0.9, die lediglich bei Synonymen in den Anfragen erreicht werden können, liegt der Faktor mit unter 20 noch in einer akzeptablen Größenordnung.

Die Nachbearbeitung einer *Disjunktion* ist noch wesentlich unkritischer: Bei Anfragen der Form $A \vee B$ werden zunächst beide Anfragen einzeln ausgeführt und im anschließenden Nachbearbeitungsschritt beide Ergebnisse additiv verknüpft. Dabei muss lediglich auf Doubletten geachtet werden, also Datensätze, die in beiden Ergebnismengen enthalten sind. Kritisch ist also nur die Schnittmenge $A \cap B$, die i.d.R. aber einen geringen Umfang hat.

Als formales Kriterium für die Höhe des Aufwands dient wieder der Quotient q, der sich in diesem Fall wie folgt berechnet:

$$q = \frac{\#betrachtete\ Datensätze}{\#auszugebende\ Datensätze}$$

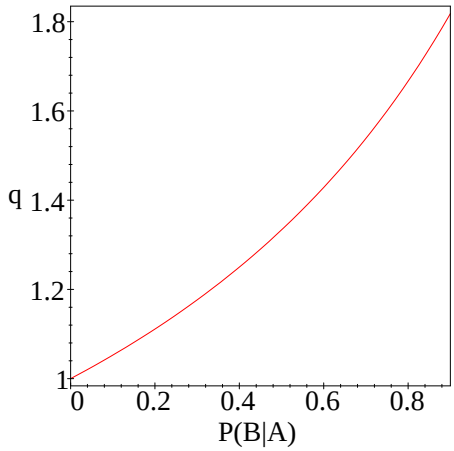


Abbildung 2.7: Aufwand für die Nachbearbeitung einer booleschen Disjunktion am Beispiel des Falls $P(A) = P(B)$. Gezeigt ist die Anzahl zu betrachtender Datensätze für einen auszugebenden Datensatz bei einer Anfrage der Form $A \vee B$ in Abhängigkeit der relativen Wahrscheinlichkeit $P(B|A)$. Der Quotient liegt mit maximal 2 in einem völlig unkritischen Bereich.

$$
\begin{aligned}
&= \frac{\#\textit{Datensätze aus } A + \#\textit{Datensätze aus } B}{\#\textit{Datensätze aus } A + \#\textit{Datensätze aus } B - \#\textit{Datensätze aus } A \cap B} \\
&= \frac{|A| + |B|}{|A| + |B| - |A \cap B|} \\
&= \frac{N * P(A) + N * P(B)}{N * P(A) + N * P(B) - N * P(A \wedge B)} \\
&= \frac{P(A) + P(B)}{P(A) + P(B) - P(A \wedge B)} \\
&= \frac{P(A) + P(B)}{P(A) + P(B) - P(B|A) * P(A)} \qquad \textit{Ann.} : P(A) = P(B) =: P \\
&= \frac{2 * P}{2 * P - P(B|A) * P} \\
&= \frac{2}{2 - P(B|A)}
\end{aligned}
$$

Dieser Quotient ist in Abb. 2.7 graphisch dargestellt, und kann offensichtlich maximal den Wert 2 erreichen. Er liegt damit in einem für die Praxis absolut unproblematischen Bereich.

2.3.2.5 Nachbildung boolescher Konjunktionen

Analog zu der Nachbildung von Disjunktionen können Konjunktionen nachgebildet werden, indem die einzelnen Anfragen ausgeführt werden, und die Ergebnismengen nachträglich verknüpft werden, indem datensatzweise vergleichen wird und nur die Datensätze ausgegeben werden, die in beiden Mengen bzw. Streams vorkommen.

Dabei erreicht der Aufwand jedoch schnell eine kritische Grenze: Untersucht man das bereits in den vorherigen Kapiteln verwendete Aufwandsmaß q, das die Anzahl zu betrachtender Datensätze für einen auszugebenden Datensatz ausdrückt, so erhält man einen im Wesentlichen hyperbolischen Zusammenhang:

$$
\begin{aligned}
q &= \frac{\#betrachtete\ Datensätze}{\#auszugebende\ Datensätze} \\
&= \frac{\#Datensätze\ aus\ A + \#Datensätze\ aus\ B}{\#Datensätze\ ausA \cap B} \\
&= \frac{|A| + |B|}{|A \cap B|} \\
&= \frac{N * P(A) + N * P(B)}{N * P(A \wedge B)} \\
&= \frac{P(A) + P(B)}{P(A \wedge B)} \\
&= \frac{P(A) + P(B)}{P(B|A) * P(A)} \qquad Ann.:\ gleiche\ Auftrittsw.\ P(A) = P(B) =: P \\
&= \frac{2 * P}{P(B|A) * P} \\
&= \frac{2}{P(B|A)}
\end{aligned}
$$

Wie man in Abbildung 2.8 erkennen kann, ist dieser Quotient aufgrund des hyperbolischen Verlaufs nicht nach oben begrenzt. D.h. für eine bedingte Wahrscheinlichkeit von z.B. 0.1% für $P(B|A)$ müssen im Mittel 2000 Datensätze von einer Datenbank angefordert (und paarweise verglichen) werden, damit ein Datensatz ausgegeben werden kann. Für noch kleinere Wahrscheinlichkeiten steigt dieser Wert entsprechend schnell an. Gerade für nicht lokal vorliegende Datenbanken - wie es bei Metasuchmaschinen häufig der Fall ist - stellt das Netzwerk den begrenzenden Faktor dar, so dass keine akzeptablen Antwortzeiten mehr erreicht werden können.

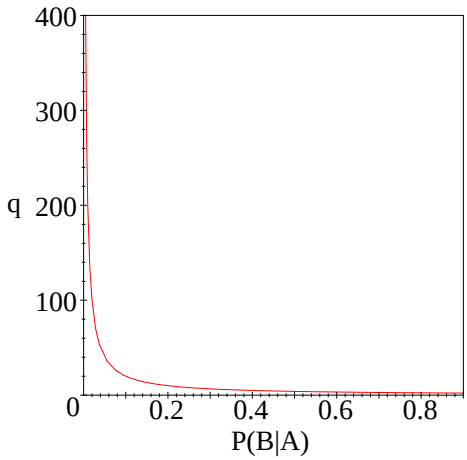


Abbildung 2.8: Aufwand für die Nachbearbeitung einer booleschen Konjunktion. Gezeigt ist die Anzahl zu betrachtender Datensätze für einen auszugebenden Datensatz bei einer Anfrage der Form $A \wedge B$ in Abhängigkeit der relativen Wahrscheinlichkeit $P(B|A)$. Aufgrund des hyperbolischen Verlaufs gibt es keine obere Grenze für der Aufwand, so dass auf eine derartige Nachbearbeitung verzichtet werden muss.

2.4 Zusammenfassung

In diesem Kapitel wurden die generellen Problemstellungen einer Fuzzy-Metasuchmaschine diskutiert. Es wurden Aufwandsabschätzungen angestellt, und die Probleme der Heterogenität der Datenquellen aufgezeigt.

Für die verschiedenen Problemstellungen wurden Lösungen vorgestellt, insbesondere wurde eine 3-Schichten- bzw. Mediator-Architektur mit dem Konzept der Wrapper, die das Problem der Heterogenität lösen, sowie das Streaming-Prinzip, welches der enormen Datenmengen gerecht wird, erläutert.

Dabei wurde insbesondere auf die verschiedenen Nachbearbeitungsschritte, die ein Wrapper auf Streams durchzuführen hat, eingegangen. Es wurde gezeigt, dass ein Wrapper durch eine intelligente Nachbearbeitung einen Großteil an mangelnder Komplexität auf Seite der Datenbank ausgleichen kann. Unvollständige Sortierung, das Fehlen von Disjunktionen und von Negationen (von Literalen) in der von der Datenbank unterstützten, booleschen Anfragesyntax müssen keine generellen Ausschlusskriterien für die Einbindung einer Datenbank in eine Metasuchmaschine sein. Lediglich bei Konjunktionen stoßen die Verfahren an ihre Grenzen.

Weitere Grenzen der Nachbearbeitung ergeben sich in der Praxis häufig dadurch, dass bestimmte Datenbanken *mehrere* der erforderlichen Features nicht unterstützen und verschiedene der in den vorherigen Kapiteln *einzeln* betrachteten Nachbearbeitungsschritte kombiniert werden müssen. Der nötige Aufwand multipliziert sich entsprechend und kann dann schnell an Grenzen stoßen, wie die folgenden Abschätzungen zeigen.

Tabelle 2.3 gibt einen Überblick über den relativen Gesamtaufwand - also die Anzahl an von der Datenbank anzufordernden Dokumente für ein auszugebendes Dokument - in Abhängigkeit der verschiedenen, zu berücksichtigenden und teilweise in den vorherigen Unterkapiteln diskutierten Faktoren. Dabei hat der Aufwand der Aggregation einen besonders großen Einfluss auf das Laufzeitverhalten. Kapitel 4 wird daher detailliert auf diesen Faktor eingehen. Darüber hinaus ist zu erkennen, dass auch bei nur mittlerem Aggregations-Aufwand die „worst case"-Kombination der anderen Faktoren zu einem hohen relativen Gesamtaufwand von $q = 4000$ führt. Da i.d.R. vom Benutzer nicht nur ein Dokument angefordert wird, sondern eine Ergebnisseite bereits ca. 25 Dokumente enthält, ergibt sich im Produkt eine Gesamtanzahl von mehreren Tausend anzufordernden Dokumenten. Dieser Wert liegt z.B. für Datenbanken, die ausschließlich über HTTP/CGI zugänglich sind, völlig außerhalb praktikabler Grenzen.

Für das Konzept einer Fuzzy-Metasuchmaschine bedeutet dies, dass eine sehr spezifische Entscheidung - teilweise auf Basis von Heuristiken - getroffen werden muss, ob die Integration einer konkreten Datenbank sinnvoll ist oder nicht: Einzelne Features können relativ problemlos durch eine entsprechende Nachbearbeitung vom Wrapper realisiert werden, jedoch kann ein *zu* hoher Mangel an Komplexität der Datenbank den nötigen Aufwand auf ein unpraktikables Maß erhöhen.

Beim Rechercheassistenten [57] wurden alle vorgestellten Verfahren der Nachbearbeitung nicht nur einzeln, sondern auch in teilweiser Kombination miteinander erfolgreich eingesetzt

und es können daher selbst unterschiedliche Internet-Datenbanken in einer Metasuche gemeinsam durchsucht werden. Dabei war die Integration unvollständig sortierter Datenbanken ebenso möglich wie die Einbindung von Datenbanken, die in ihrer Anfragesyntax sehr restriktiv sind und lediglich Konjunktionen einzelner Stichworte erlauben.

Aufwand für Aggregation	Nachsortierung (Ausreisserw. p)	Negation/Disjunktion (bed. W. $P(B\|A)$)	Gesamtaufwand (q)
10	0.01	0.1	10
10	0.01	0.5	40
10	0.10	0.1	10
10	0.10	0.5	40
10	0.60	0.1	100
10	0.60	0.5	400
100	0.01	0.1	100
100	0.01	0.5	400
100	0.10	0.1	100
100	0.10	0.5	400
100	0.60	0.1	1000
100	0.60	0.5	4000
1000	0.01	0.1	1000
1000	0.01	0.5	4000
1000	0.10	0.1	1000
1000	0.10	0.5	4000
1000	0.60	0.1	10000
1000	0.60	0.5	40000

Tabelle 2.3: Abschätzung des Gesamtaufwandes für kombinierte Nachbearbeitungsschritte, gemessen in Anzahl zu betrachtender Datensätze für die Berechnung eines Ergebnis-Datensatzes (Der Aufwand für die Aggregation ist sehr unterschiedlich, vgl. Kapitel 4.)

3

Fuzzy-Operatoren für bibliographisches Retrieval

3.1 Motivation und Kapitelübersicht

Bei Fuzzy-Operatoren wird im Folgenden zwischen „Filter-Operatoren" (bzw. Zugehörigkeitsfunktionen) und „Aggregations-Operatoren" (auch „Quantoren") unterschieden. Filter-Operatoren sind Operatoren, die einer Entität einen Zugehörigkeitsgrad zu einer Fuzzy-Menge zuordnen (z.B. „Datum möglichst neu"). Aggregations-Operatoren verknüpfen mehrere Fuzzy-Mengen (z.B. Mengen von Dokumenten, die bestimmte Stichworte enthalten) zu einer Fuzzy-Menge, und korrespondieren somit mit linguistischen Quantoren wie „einige", „viele" oder „alle". Für die Definition intuitiv plausibler Fuzzy-Operatoren sind (u.a.) zwei Kriterien wichtig für das gewichtete Retrieval:

- **Einfluss:**
  - Jede semantische Änderung in der Anfrage soll auch Einfluss auf die Relevanz eines Dokumentes haben.
  - Zwei Dokumente, die sich in einem (in der Anfrage spezifizierten) Attribut unterscheiden, sollen unterschiedliche Relevanzen bekommen.
- **Stabilität:**
  - *Geringfügige* semantische Änderungen in der Anfrage sollen auch nur *geringfügige* Unterschiede im Ergebnis bewirken.
  - Dokumente, die sich nur geringfügig unterscheiden, sollen ähnliche Relevanzen erhalten.

Diese beiden Kriterien werden in den Kapitel 3.2 und 3.3 für Filter- bzw. Aggregations-Operatoren formalisiert und verfeinert, und es werden spezifische Eigenschaften abgeleitet, die eine optimale Auswahl von Fuzzy-Operatoren für das bibliographische Retrieval ermöglichen.

In Kapitel 3.4 werden darauf aufbauend Anfrageoptimierungen vorgestellt, die äquivalente Umformungen von Fuzzy-Anfragen ermöglichen, gleichzeitig aber zu einer effizienteren Abarbeitung führen.

3.2 Zugehörigkeitsfunktionen, Filter-Operatoren

„Filter-Operatoren" sind definiert durch Zugehörigkeitsfunktionen, also Abbildungen, die einer Entität anhand eines Attributes den *Zugehörigkeitsgrad* μ zu einer Fuzzy-Menge zuordnen. Beispielsweise teilt ein Operator „möglichst neu" (vgl. Abbildung 3.1) einem 10 Jahre alten Dokument einen mittleren Zugehörigkeitsgrad von etwa $\mu_{moeglichst\,neu} = 0.5$ zu. Dieser Filter-Operator arbeitet also auf der *linguistischen Variable* „Alter" bzw. „Erscheinungsdatum" und implementiert die *Zugehörigkeitsfunktion* μ zu dem *linguistischen Term* „möglichst neu" (vgl. beispielsweise [42]). Im folgenden wird, falls nicht explizit anders erwähnt, davon ausgegangen, dass die von einem Filter-Operator gelieferte Relevanz multiplikativ mit der Original-Relevanz eines Datensatzes verknüpft wird.

Definition 3.1: Filter-Operator

Ein Filter-Operator, der eine Fuzzy-Menge festlegt, ist definiert durch eine Zugehörigkeitsfunktion μ, die den Elementen a eines Universums A einen Zugehörigkeitsgrad zwischen 0 und 1 zuordnet:

$$\mu_{<linguistischer\,Term>} : A \rightarrow [0..1]$$
$$\mu = \mu_{<linguistischer\,Term>}(a) \quad a \in A$$

Die wichtigsten Eigenschaften von Filter-Operatoren bzw. Zugehörigkeitsfunktionen, die in den weiteren Kapiteln (insbesondere für Anfrageoptimierungen) häufiger benötigt werden, sind die folgenden:[1]

Definition 3.2: α-Schnitt, strenger α-Schnitt

Der α-*Schnitt* $F_{\geq\alpha}$ einer Fuzzy-Menge F über dem Universum A ist definiert als:

$$F_{\geq\alpha} := \{x \in A | \mu_F(x) \geq \alpha\}$$

Analog ist der *strenge* α-*Schnitt* $F_{>\alpha}$ definiert als:

$$F_{>\alpha} := \{x \in A | x\mu_F(x) > \alpha\}$$

[1] Für eine detailliertere Abhandlung sei auf die entsprechende Literatur (z.B. [42, 73]) verwiesen.

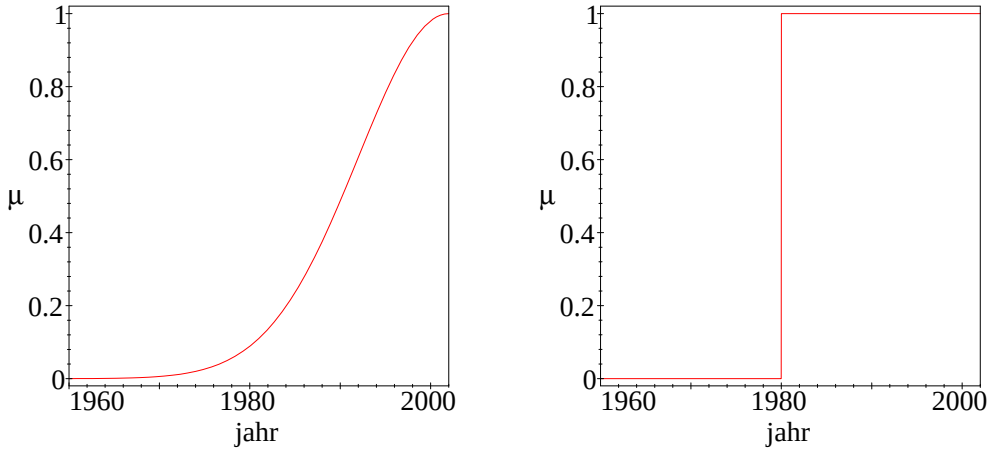


Abbildung 3.1: Fuzzy bzw. boolesche Zugehörigkeitsfunktionen der Filter-Operatoren „möglichst neu" (links) und „nach 1980" (rechts). Die nicht-stetige, boolesche Zugehörigkeitsfunktion erlaubt keine feine Granulierung der Relevanzen.

Der (strenge) α-Schnitt ist also eine klassische, boolesche Menge all derjenigen Elemente einer Fuzzy-Menge, deren Zugehörigkeitsgrad größer/gleich (bzw. echt größer als) ein festgelegter Grenzwert ist.

Definition 3.3: Boolesche Hülle

Die *boolesche Hülle* (auch *envelope*, *support* oder *Trägermenge*) $env(F)$ einer Fuzzy-Menge F über dem Universum A ist definiert als:

$$env(F) := \{x \in A | \mu_F(x) > 0\}$$

Die boolesche Hülle ist also die klassische, boolesche Menge aller Elemente der Fuzzy-Menge, deren Zugehörigkeitsgrad größer ist als 0, also ein Spezialfall des strengen α-Schnittes für $\alpha = 0$.

Definition 3.4: Höhe, Normalisierte Fuzzy-Menge

Die *Höhe* $H(F)$ einer Fuzzy-Menge F ist definiert als das Supremum ihrer Zugehörigkeitsfunktion:

$$H(F) := \sup_x(\mu_F(x))$$

Ist die Höhe einer Fuzzy-Menge 1, so wird die Fuzzy-Menge *normalisiert* genannt.

Definition 3.5: Fuzzy-Zahl

Eine Fuzzy-Menge heißt *Fuzzy-Zahl*, genau dann wenn ihre Trägermenge die Menge der reellen Zahlen ist und sie normalisiert ist.

Eine für das Fuzzy-Retrieval wichtige Eigenschaft ist die Stetigkeit. Durch sie wird die eingangs erläuterte Stabilität sichergestellt: Dokumente, die z.B. ein ähnliches Erscheinungsdatum haben, bekommen von einem stetigen Filter-Operator eine ähnliche Relevanz zugewiesen:

Definition 3.6: Stetigkeit

Eine Fuzzy-Menge/ein Filter-Operator F, auf deren Trägermenge eine natürliche Ordnung definiert ist, ist genau dann stetig, wenn ihre Zugehörigkeitsfunktion stetig ist:

$$\forall\delta\exists\epsilon: \quad |x-y|<\epsilon \quad \Rightarrow \quad |\mu_F(x)-\mu_F(y)|<\delta$$

Eine weitere wichtige Eigenschaft von Filter-Operatoren ist die strenge Monotonie, durch die der *Einfluss* sichergestellt wird: Ein streng monotoner Filter-Operator gewährleistet eine feine Granulierung der Relevanzen verschiedener Dokumente. Bei einem lediglich schwach monotonen - oder gar booleschen - Operator wie „nach 1980” (vgl. Abb. 3.1) ist dies nicht gegeben - ein Dokument aus dem Jahr 1981 bekäme genau wie ein Dokument aus dem Jahr 2000 eine Relevanz von 1.0:

Definition 3.7: Monotonie

Eine Fuzzy-Menge mit einer Trägermenge F, auf der eine natürliche Ordnung definiert ist, ist *monoton steigend*, genau dann wenn ihre Zugehörigkeitsfunktion monoton steigend ist:

$$x<x' \Rightarrow \mu_F(x)\leq\mu_F(x') \qquad \forall x,x'\in F$$

Analog ist eine Fuzzy-Menge *monoton fallend*, genau dann wenn ihre Zugehörigkeitsfunktion monoton fallend ist:

$$x<x' \Rightarrow \mu_F(x)\geq\mu_F(x') \qquad \forall x,x'\in F$$

Sie ist *streng monoton steigend*, genau dann wenn ihre Zugehörigkeitsfunktion streng monoton steigend ist:

$$x<x' \Rightarrow \mu_F(x)<\mu_F(x') \qquad \forall x,x'\in F$$

Analog ist sie *streng monoton fallend*, genau dann wenn ihre Zugehörigkeitsfunktion streng monoton fallend ist:

$$x<x' \Rightarrow \mu_F(x)>\mu_F(x') \qquad \forall x,x'\in F$$

3.3 Aggregations-Operatoren (Quantoren)

Als „Aggregations-Operator" (auch „Quantor") wird im Folgenden ein Operator bezeichnet, der zwei oder mehr Fuzzy-Mengen miteinander verknüpft. Er stellt damit eine Verallgemeinerung der bekannten Konzepte der Vereinigungs- und Schnittmengen klassischer, boolescher Mengen auf unscharfe Fuzzy-Mengen dar. Bei dieser Verknüpfung ist es sinnvoll, die Zugehörigkeitsgrade zusätzlich mit *Gewichten* versehen zu können. Bei der Formulierung einer Anfrage kann der Benutzer dem System so mitteilen, wie *wichtig* ihm die einzelnen Suchbegriffe sind.

Es ist also zu unterscheiden zwischen dem *Zugehörigkeitsgrad* $\mu_a(d)$ eines Dokumentes d zu der Menge an Dokumenten, die ein Stichwort a enthält (der sog. „Term-Dokument-Relevanz", die bereits durch die Indexierung der Datenbank festgelegt wird) und dem vom Benutzer zugeteilten Gewicht w_a dieses Stichwortes (der „Term-Benutzer-Relevanz"). So erhält ein Dokument, welches hauptsächlich die vom Benutzer als *wichtig* eingestuften Stichworten enthält, eine höhere Relevanz als ein Dokument, welches vorwiegend als *weniger wichtig* gekennzeichnete Stichworte beinhaltet.

Den Gewichten kommt eine besondere Bedeutung zu, da die Zugehörigkeitsgrade der Dokumente zu Dokumentenmengen, die ein bestimmtes Stichwort enthalten, aufgrund der üblichen Indexierungsverfahren vieler Datenbanken als Spezialfall häufig boolesch sind. Dies muss jedoch nicht zwangsläufig der Fall sein, z.B. wenn die Datenbank auch eine fehlertolerante Suche durch approximatives Matching, Wortstammbildung, Kompositazerlegung oder ähnliche Verfahren ermöglicht oder wenn - wie z.B. beim Rechercheassistenten [57] - ein Begriff, der nur im Abstract oder Volltext eines Dokumentes vorkommt, einen niedrigeren Zugehörigkeitsgrad erhalten soll als ein Stichwort, welches im Titel des Dokumentes auftaucht.

Definition 3.8: Aggregations-Operator

Ein *Aggregations-Operator* ist definiert durch eine n-stellige Funktion μ_{agg}, die verschiedene Zugehörigkeitsgrade $\mu_{F_i}(a)$, die eine Entität a zu verschiedenen Fuzzy-Mengen F_i hat, zu der Relevanz einer „Vereinigung" $F_1 \diamond F_2 \diamond ... \diamond F_n$ dieser Mengen verknüpft:

$$\mu_{agg} : [0..1]^n \rightarrow [0..1]$$
$$\mu_{F_1 \diamond F_2 \diamond ... \diamond F_n}(a) = \mu_{agg}(\mu_{F_1}(a), ..., \mu_{F_n}(a))$$

Analog ist ein *gewichteter Aggregations-Operator* definiert durch eine n-stellige Funktion μ_{agg}, die verschiedene mit Gewichten w_i versehene Zugehörigkeitsgrade $\mu_{F_i}(a)$, die eine Entität a zu verschiedenen Fuzzy-Mengen F_i hat, zu der Relevanz einer „Vereinigung" $F_1 \diamond F_2 \diamond ... \diamond F_n$ dieser Mengen verknüpft:

$$\mu_{agg} : \langle [0..1], [0..1] \rangle^n \rightarrow [0..1]$$
$$\mu_{F_1 \diamond F_2 \diamond ... \diamond F_n}(a) = \mu_{agg}(\langle w_1, \mu_{F_1}(a) \rangle ... \langle w_n, \mu_{F_n}(a) \rangle)$$

Es stellt sich zwangsläufig die Frage, ob die Tupel aus Gewicht und Zugehörigkeitsgrad nicht im Voraus und *unabhängig* vom Aggregations-Operator mit einer Gewichtungsfunktion $g(w_i, \mu_i)$ verknüpft werden können und das Resultat an den Aggregations-Operator weitergegeben werden kann, um den Aggregations-Operator ausschließlich von Skalaren abhängig zu machen und damit zu vereinfachen (siehe z.B. [72]):

$$\mu = \mu_{agg}(g_1, ..., g_n) \quad \text{mit} \quad g_i = g(w_i, \mu_i)$$

Der $\Sigma Count$-Ansatz, sowie andere Ansätze, die auf Kardinalitätsmaßen von Fuzzy-Mengen basieren und die in Kapitel 3.3.2.3 vorgestellt werden, schlagen sogar vor, den Aggregations-Operator nur von einem einzigen Skalar abhängig zu machen, und *sämtliche* Gewichte und Zugehörigkeitsgrade im Voraus zu verknüpfen:

$$\mu_{agg} = \mu_{agg}(f) \quad \text{mit} \quad f = f(< w_1, \mu_1 >, ..., < w_n, \mu_n >)$$

Derartige Vereinfachungen können jedoch *nicht* konsistent zu Ende geführt werden, wie für den Grenzwert $w = 0$ deutlich wird: Wird einem Stichwort ein Gewicht von $w = 0$ zugewiesen, so bedeutet dies, dass das Stichwort irrelevant für die Anfrage ist, und damit keinen Einfluss auf das Gesamtergebnis haben soll. Die Gewichtungsfunktion g bzw. das Kardinalitätsmaß f muss für diesen Fall also das *neutrale Element* für die Aggregation μ_{agg} liefern. Allein für die booleschen Spezialfälle UND und ODER sind die neutralen Elemente jedoch unterschiedlich: Das neutrale Element des UND-Operators ist 1, das des ODER-Operators ist 0. Zwar wird der Grenzfall $w = 0$ in der Praxis nicht vorkommen, die Argumentation greift jedoch auch für Werte *nahe 0*, also kleine w: in diesen Fällen muss, um die Stabilität zu bewahren, analog ein Wert zurückgegeben werden, der nah an dem des neutralen Elements liegt. Aus diesem Grund kann ein Operator, der auf diese Weise definiert ist, auch nicht die im weiteren gestellte Parametrisierungs-Forderung erfüllen. Glöckner und Knoll stellen diese und weitere Inkonsistenzen, die sich durch einen derartigen Ansatz ergeben, z.B. in [25, 27] dar.

Theorem 3.1: Inkonsistenz von Kardinalitätsansätzen

Es ist nicht möglich, eine Klasse von Fuzzy-Aggregationsoperatoren bzw. Fuzzy-Quantoren

$$\mu_{agg} : \langle [0..1], [0..1] \rangle^n \rightarrow [0..1]$$
$$\mu = \mu_{agg}(\langle w_1, \mu_1 \rangle ... \langle w_n, \mu_n \rangle)$$

über eine einheitliche Gewichtungsfunktion g bzw. f mit

$$\mu = \mu_{agg}(g_1, ..., g_n) \quad \text{mit} \quad g_i = g(w_i, \mu_i)$$

bzw.

$$\mu = \mu_{agg}(f) \quad \text{mit} \quad f = f(< w_1, \mu_1 >, ..., < w_n, \mu_n >)$$

zu definieren, wenn UND- und ODER-Operator eingeschlossen werden sollen.

3.3.1 Kriterien für die bibliographische Suche

Im folgenden werden z.T. aus der Literatur (vgl. z.B. Waller-Kraft-Wish-List [68] oder [12, 42, 73]) bekannte Eigenschaften und Anforderungen an Aggregations-Operatoren vorgestellt und ihre Bedeutung für das bibliographische retrieval diskutiert. In den nachfolgenden Kapiteln werden verschiedene Operatoren auf diese Eigenschaften hin untersucht.

Eine sinnvollerweise zu fordernde Eigenschaft ist die Kommutativität: Die Reihenfolge, in der ein Benutzer Stichworte in ein Retrieval-System eingibt, soll keine Rolle spielen (vorausgesetzt, die mit den Stichworten korrespondierenden Gewichte werden ebenfalls vertauscht):

Definition 3.9: Kommutativität

Ein Aggregations-Operator ist kommutativ, wenn gilt:

$$\mu_{agg}(\mu_1, ..., \mu_n) = \mu_{agg}(\mu_{j_1}, ..., \mu_{j_n})$$

bzw.

$$\mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) = \mu_{agg}(\langle w_{j_1}, \mu_{j_1}\rangle, ..., \langle w_{j_n}, \mu_{j_n}\rangle)$$

für jede Permutation $(j_1, ..., j_n)$.

Eine in der Literatur an einige Quantoren gestellte Forderung ist die Assoziativität:

Definition 3.10: Assoziativität

Ein ungewichteter Aggregations-Operator ist assoziativ, wenn gilt:

$$\mu_{agg}(\mu_{agg}(\mu_a, \mu_b), \mu_c) = \mu_{agg}(\mu_a, \mu_b, \mu_c) = \mu_{agg}(\mu_a, \mu_{agg}(\mu_b, \mu_c))$$

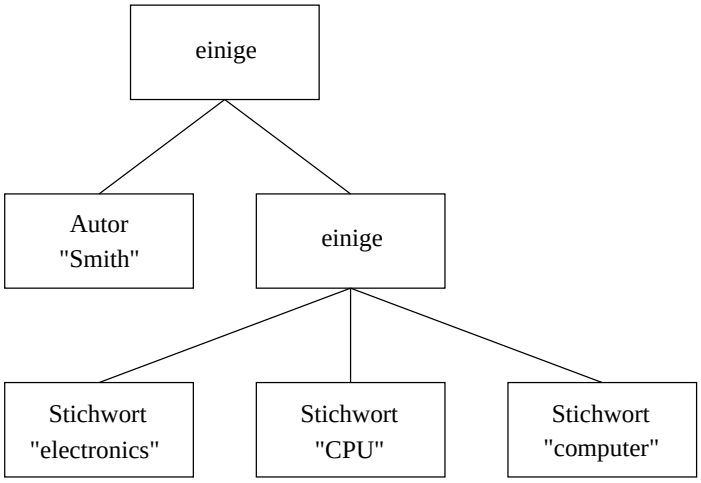
Ein gewichteter Aggregations-Operator ist assoziativ, wenn gilt:

$$\begin{aligned} & \mu_{agg}(\langle w_{a,b}, \mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle)\rangle, \langle w_c, \mu_c\rangle) \\ = \; & \mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle, \langle w_c, \mu_c\rangle) \\ = \; & \mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_{b,c}, \mu_{agg}(\langle w_b, \mu_b\rangle, \langle w_c, \mu_c\rangle)\rangle) \end{aligned}$$

Im Gegensatz zu den booleschen Spezialfällen UND und ODER spielt die Assoziativität beim gewichteten Retrieval eine untergeordnete Rolle und kann sich in einigen Fällen auf den gewünschten Suchprozess eher negativ auswirken. (So erscheint bereits die formale Notwendigkeit der neuen Gewichte $w_{a,b}$ und $w_{b,c}$ wenig intuitiv.) Das folgende Beispiel verdeutlicht dies:

Dem Assoziativgesetz nach sind die beiden Anfragen aus Abbildung 3.2 äquivalent. Dies entspricht jedoch nicht dem intuitiven Verständnis: Während in der ersten Anfrage dem Autor eine hervorgehobene Rolle zukommt, ist dies in Anfrage (2) nicht mehr erkennbar. Dort ist der Autor ein gleichrangiger Suchbegriff unter vielen. Für einige Spezialfälle von

Anfrage 1:



Anfrage 2:

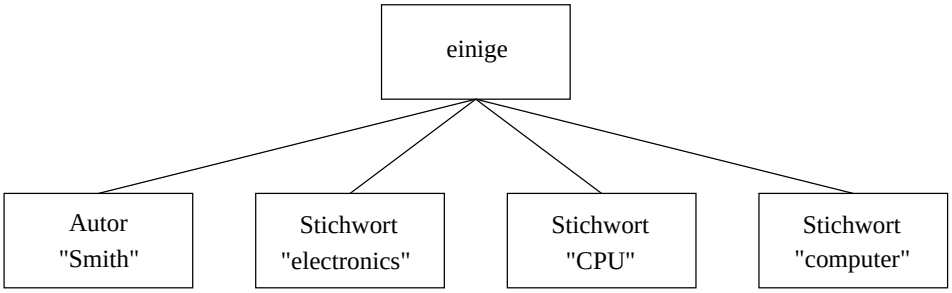


Abbildung 3.2: Zwei nach dem Assoziativgesetz äquivalente Anfragebäume. Eine Übertragung der Forderung nach Assoziativität von den booleschen Operatoren auf Fuzzy-Quantoren ist nicht generell sinnvoll: Während in Anfrage (1) der Autor eine besondere Rolle einnimmt, ist er in Anfrage (2) ein gleichwertiger Suchbegriff unter vielen.

Aggregations-Operatoren, wie die klassischen Konjunktionen und Disjunktionen ist die Assoziativitäts-Forderung hingegen sinnvoll.

Die Forderung nach Monotonie in den verschiedenen Parametern kommt der eingangs formulierten Forderung des Einflusses nach:

Definition 3.11: ∃-Monotonie bezüglich der Zugehörigkeitsgrade

Ein Aggregations-Operator ist schwach ∃-monoton steigend bezüglich der Zugehörigkeitsgrade, wenn gilt:

$$(\exists i : \mu_i < \mu_i') \quad \Rightarrow \quad \mu_{agg}(..., \langle w_i, \mu_i \rangle, ...) \leq \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_i, \mu_i' \rangle, ..., \langle w_n, \mu_n \rangle)$$

Ein Aggregations-Operator ist streng ∃-monoton steigend bezüglich der Zugehörigkeitsgrade, wenn gilt:

$$(\exists i : \mu_i < \mu_i') \quad \Rightarrow \quad \mu_{agg}(..., \langle w_i, \mu_i \rangle, ...) < \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_i, \mu_i' \rangle, ..., \langle w_n, \mu_n \rangle)$$

Wichtig ist hier der Unterschied zu der in der Literatur oft nicht explizit herausgestellten oder gar wesentlich schwächer formulierten Forderung, dass bereits bei der Existenz *eines* abweichenden Zugehörigkeitsgrades (anstatt aller) die Relevanz beeinflusst wird, weswegen hier explizit der Begriff ∃-Monotonie verwendet wird: (Vgl. z.B. [21], wonach μ_{agg} streng monoton ist, wenn $(\forall i : \mu_i < \mu_i') \Rightarrow \mu_{agg}(\mu_1, ..., \mu_n) < \mu_{agg}(\mu_1', ..., \mu_n')$.) Nach dieser schwächeren Forderung wäre z.B. auch die min-Norm, die der Forderung das Einflusses nicht nachkommt, streng monoton.

Offensichtlich ist es für das bibliographische Retrieval sinnvoll, von einem Aggregations-Operator die *strenge* ∃-Monotonie zu fordern, damit eine möglichst feine Granulierung der Relevanz bezüglich einer Anfrage - und damit eine gute Unterscheidbarkeit der Dokumente sichergestellt ist. Es wird gefordert, dass ein Dokument, das einen höheren Zugehörigkeitsgrad zu einem Stichwort aufweist als ein bzgl. aller anderen Parameter identisches Dokument, eine höhere Gesamt-Relevanz bekommt.

Monotonie wird ebenfalls für die Gewichte gefordert:

Definition 3.12: ∃-Monotonie bezüglich der Gewichte

Ein Aggregations-Operator ist schwach ∃-monoton bezüglich der Gewichte, wenn gilt:

$$w_n < w_n' \quad \Rightarrow \quad \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_n, \mu_n \rangle) \leq \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_n', \mu_n \rangle)$$
$$\text{für } \mu_n > \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_{n-1}, \mu_{n-1} \rangle)$$
$$w_n < w_n' \quad \Rightarrow \quad \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_n, \mu_n \rangle) \geq \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_n', \mu_n \rangle)$$
$$\text{für } \mu_n < \mu_{agg}(\langle w_1, \mu_1 \rangle, ..., \langle w_{n-1}, \mu_{n-1} \rangle)$$

Ein Aggregations-Operator ist streng ∃-monoton bezüglich der Gewichte, wenn gilt:

$$w_n < w'_n \quad \Rightarrow \quad \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) < \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w'_n, \mu_n\rangle)$$
$$\text{für } \mu_n > \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_{n-1}, \mu_{n-1}\rangle)$$
$$w_n < w'_n \quad \Rightarrow \quad \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) > \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w'_n, \mu_n\rangle)$$
$$\text{für } \mu_n < \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_{n-1}, \mu_{n-1}\rangle)$$

Die ∃-Monotonie bezüglich der Gewichte hat eine vergleichbare Bedeutung wie die ∃-Monotonie bezüglich der Zugehörigkeitsgrade, jedoch sind hier die Voraussetzungen komplexer: Es wird gefordert, dass bei Eingabe eines *zusätzlichen* Suchbegriffs, der im Dokument tatsächlich vorkommt, die Relevanz des Dokumentes monoton mit dem Gewicht ansteigt, und bei Eingabe eines Suchbegriffs, der *nicht* im Dokument vorkommt, die Relevanz monoton fällt, wenn das korrespondierende Gewicht erhöht wird. Der „Grenzwert" für den Zugehörigkeitsgrad ist dabei die Gesamt-Relevanz des Dokumentes bezüglich der Anfrage *ohne* das entsprechende Stichwort.

Definition 3.13: Korrelierte Monotonie

Ein Aggregations-Operator ist schwach k-monoton, wenn gilt:

$$\mu_k < \mu_l \;\Rightarrow\; \mu_{agg}(..., \langle w_k, \mu_k\rangle, \langle w_l, \mu_l\rangle, ...) \leq \mu_{agg}(..., \langle w_k - \epsilon, \mu_k\rangle, \langle w_l + \epsilon, \mu_l\rangle, ...)$$
$$\forall \epsilon > 0$$

Ein Aggregations-Operator ist streng k-monoton, wenn gilt:

$$\mu_k < \mu_l \;\Rightarrow\; \mu_{agg}(..., \langle w_k, \mu_k\rangle, \langle w_l, \mu_l\rangle, ...) < \mu_{agg}(..., \langle w_k - \epsilon, \mu_k\rangle, \langle w_l + \epsilon, \mu_l\rangle, ...)$$
$$\forall \epsilon > 0$$

Die K-Monotonie hat schwächere Voraussetzungen als die anderen Monotonien, und ist damit eine noch stärkere Forderung: Sie besagt, dass die Erhöhung eines Gewichtes die Gesamt-Relevanz auch dann erhöht, wenn gleichzeitig ein anderes Gewicht verringert wird, unter der Voraussetzung, dass der Zugehörigkeitsgrad des ersten Suchbegriffs höher ist als das des zweiten Suchbegriffs. Kommt in einem Dokument beispielsweise der Begriff „retrieval" mit einem Zugehörigkeitsgrad von $\mu_{retrieval} = 1.0$ vor, der gesuchte Begriff „fuzzy" jedoch nur mit einem mittleren Zugehörigkeitsgrad von $\mu_{fuzzy} = 0.5$, dann soll die Erhöhung des Gewichts $w_{retrieval}$ die Relevanz erhöhen, auch dann, wenn das Gewicht für den - mit geringerem Zugehörigkeitsgrad vorkommenden - Begriff „fuzzy" (w_{fuzzy}) gleichzeitig vermindert wird.

Die Monotonie-Eigenschaften sollen für *alle* Gewichte und Zugehörigkeitsgrade gelten, und nicht nur für die höchsten bzw. niedrigsten, wie es bei einigen s- und t-Normen (vgl. Kapitel 3.3.2.1) der Fall ist: Auch die Änderung eines *mittleren* Gewichtes bzw. *mittleren* Zugehörigkeitsgrades soll die Gesamt-Relevanz des Dokumentes in beschriebener Weise beeinflussen. Ein ähnliches Kriterium wird z.B. in [46] mit dem „single operand dependance problem"

formuliert, das dann erfüllt ist, wenn der Aggregations-Operator immer den Wert eines der Zugehörigkeitsgrade μ_i - unabhängig vom Wert der anderen Zugehörigkeitsgrade $\mu_{j\neq i}$ - einnimmt, wie es z.B. bei der Minimums-Norm der Fall ist.

Eine weitere Forderung, die die Stabilität von Aggregations-Operatoren sicherstellt, ist die Forderung nach Stetigkeit:

Definition 3.14: Stetigkeit

Ein Aggregations-Operator ist stetig in den Gewichten, wenn gilt:

$$\forall\epsilon\exists\delta \quad |w_i - w_i'| < \delta \Rightarrow |\mu_{agg}(..., \langle w_i, \mu_i\rangle, ...) - \mu_{agg}(..., \langle w_i', \mu_i\rangle, ...)| < \epsilon$$

Ein Aggregations-Operator ist stetig in den Zugehörigkeitsgraden, wenn gilt:

$$\forall\epsilon\exists\delta \quad |\mu_i - \mu_i'| < \delta \Rightarrow |\mu_{agg}(..., \langle w_i, \mu_i\rangle, ...) - \mu_{agg}(..., \langle w_i, \mu_i'\rangle, ...)| < \epsilon$$

Aggregations-Operatoren, die die Stetigkeitsbedingungen erfüllen, zeichnen sich dadurch aus, dass kleine Änderungen an den Gewichten bzw. Zugehörigkeitsgraden auch nur kleine Änderungen in der Gesamt-Relevanz des Dokumentes nach sich ziehen.

Definition 3.15: Grenzbedingungen, neutrales Element

Für Aggregations-Operatoren werden die folgenden Grenzbedingungen gefordert:

$$\mu_{agg}(\langle w_1, 1\rangle, ..., \langle w_n, 1\rangle) = 1 \quad (1)$$
$$\mu_{agg}(\langle w_1, 0\rangle, ..., \langle w_n, 0\rangle) = 0 \quad (2)$$
$$\mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle, \langle 0, \mu_{n+1}\rangle) = \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) \quad (3)$$

Nach (3) ist $\langle 0, \mu\rangle$ ein *neutrales Element* für den Aggregations-Operator.

Für ungewichtete s- und t-Normen, also Aggregations-Operatoren, die die linguistischen Quantoren „UND" bzw. „ODER" implementieren, gelten darüber hinaus speziell:

$$\mu_{ODER}(\mu, 0) = \mu_{ODER}(0, \mu) = \mu \quad (4)$$
$$\mu_{UND}(\mu, 1) = \mu_{UND}(1, \mu) = \mu \quad (5)$$

Damit ist 0 ein neutrales Element für ODER, und 1 ein neutrales Element für UND. Speziell folgt weiterhin:

$$\mu_{ODER}(\mu, 1) = \mu_{ODER}(1, \mu) = 1 \quad (6)$$
$$\mu_{UND}(\mu, 0) = \mu_{UND}(0, \mu) = 0 \quad (7)$$

Allgemein wird analog zu (4)-(7) für Aggregations-Operatoren gefordert:

$$\min_i\{\mu_i\} \leq \mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) \leq \max_i\{\mu_i\} \quad (8)$$

Die Bedeutung der Grenzbedingungen für das gewichtete Retrieval sind die folgenden:

1. Ein Dokument, in dem *alle* gewünschten Suchbegriffe vorkommen, erhält die maximale Relevanz 1.0.
2. Ein Dokument, in dem *keiner* der eingegebenen Suchbegriffe vorkommt, erhält eine Relevanz von 0.0.
3. Ein Suchbegriff mit einem Gewicht von 0.0 hat keinen Einfluss auf das Ergebnis der Anfrage.

Die letzte Forderung ist v.a. sinnvoll in Zusammenhang mit der Forderung nach Stetigkeit und drückt zusammen mit dieser aus, dass das Vorhandensein von einem Suchbegriff in einem Dokument, welcher ein kleines Gewichte hat, nur einen kleinen Einfluss auf die Gesamt-Relevanz des Dokumentes hat.

Die spezielleren Grenzbedingungen (4)-(8) an den UND- bzw. ODER-Operator sind in der Literatur bereits ausgiebig diskutiert.

Aus den o.g. Bedingungen lässt sich zudem die *Distributivität* von Aggregations-Operatoren unter bestimmten Voraussetzungen herleiten, die in den folgenden Kapiteln benötigt wird:

Theorem 3.2: Partielle Distributivität

Sei μ_{agg} ein Aggregations-Operator, s eine s- oder t-Norm (also eine Implementierung der linguistischen Quantoren „UND" bzw. „ODER") und μ_c boolesch. Dann gilt das Distributivgesetz:

$$s(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), \mu_c) = \mu_{agg}(\langle w_a, s(\mu_a, \mu_c)\rangle, \langle w_b, s(\mu_b, \mu_c)\rangle)$$

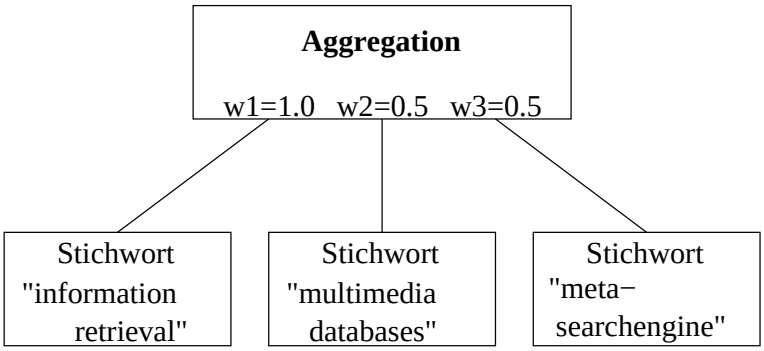
Der Beweis wird in Anhang B.1 erbracht. Die partielle Distributivität wird im Folgenden für diverse Optimierungen benötigt.

Definition 3.16: Parametrisierter Aggregations-Operator

Ein Aggregations-Operator ist *parametrisiert*, wenn er von einem weiteren Parameter λ abhängt. Von ihm wird gefordert:

$$\begin{aligned}\exists\lambda_1,\lambda_2\forall w: \quad & \mu_{agg\lambda_1}(\langle w,1\rangle, \underbrace{\langle\frac{w}{n},0\rangle,...,\langle\frac{w}{n},0\rangle}_{n-mal}) < \mu_{agg\lambda_1}(\langle w,0\rangle, \underbrace{\langle\frac{w}{n},1\rangle,...,\langle\frac{w}{n},1\rangle}_{n-mal}) \\ & \wedge \\ & \mu_{agg\lambda_2}(\langle w,1\rangle, \underbrace{\langle\frac{w}{n},0\rangle,...,\langle\frac{w}{n},0\rangle}_{n-mal}) > \mu_{agg\lambda_2}(\langle w,0\rangle, \underbrace{\langle\frac{w}{n},1\rangle,...,\langle\frac{w}{n},1\rangle}_{n-mal})\end{aligned}$$

für $n \geq 2$



Doc1: "Fuzzy sets in information retrieval"

Doc2: "A meta−searchengine for multimedia databases"

Abbildung 3.3: Beispiel einer Anfrage mit parametrisiertem Fuzzy-Quantor: Je nach ausgewähltem Aggregations-Operator kann entweder Dokument (1) oder Dokument (2) bevorzugt werden: Ein Operator mit einer hohen Orness würde Dokument (1) eine höhere Relevanz zuweisen, da in diesem der wichtigste Suchbegriff mit dem höchsten Gewicht vorkommt. Ein Operator mit einer niedrigen Orness würde Dokument (2) eine höhere Gesamt-Relevanz zuordnen, da in diesem mehr Suchbegriffe vorkommen - wenngleich diese Suchbegriffe als weniger wichtig deklariert wurden.

Der Sinn der Parametrisierung eines Aggregations-Operators ist es, zwischen zwei - intuitiv plausiblen, aber dennoch widersprüchlichen - Aspekten, die je nach Anfrage bzw. Intention des Benutzers als unterschiedlich wichtig erachtet werden können, zu vermitteln:

- In den gelieferten Dokumenten sollen möglichst alle eingegebenen Suchbegriffe auftauchen
- In den gelieferten Dokumenten sollen möglichst die Suchbegriffe, die mit hohen Gewichten versehen sind, auftauchen

Durch den Parameter λ wird eine *natürliche Ordnung* der Klasse von Aggregations-Operatoren impliziert, die von „mindestens einer” (entsprechend ODER) über „einige”, „mehrere” und „viele” bis hin zu „alle” (entsprechend UND) geht, wobei UND in maximaler Weise den erstgenannten Aspekt unterstützt, und ODER den zweiten (vgl. Abb. 3.3). In der Literatur (z.B. [70]) wird eine ähnliche Größe - jedoch meistens nur als messbare Größe eines Operators, und nicht als explizit anzugebender, konstruktiver Parameter definiert - auch als „Or-ness” bezeichnet.

3.3.2 Analyse von Fuzzy-Quantoren

In diesem Kapitel werden verschiedene, z.T. aus der Literatur bekannte Klassen von Fuzzy-Quantoren vorgestellt und ihre Eignung für das gewichtete, bibliographische Retrieval untersucht.

3.3.2.1 t- und s-Normen

t-Normen und *s-Normen* (auch *t-Konorm*), die einer fuzzy Interpretation der booleschen Operatoren ODER bzw. UND entsprechen, und somit die Vereinigungsmenge bzw. Schnittmenge definieren, sind die in der Literatur am häufigsten anzutreffenden Beispiele von Fuzzy-Aggregationsoperatoren. Sie erlauben keine Gewichtung, sondern operieren ausschließlich auf den Zugehörigkeitsgraden μ_i (vgl. [42, 73]).

T- und s-Normen erfüllen die Forderungen nach Kommutativität (Def. 3.9), Assoziativität (Def. 3.10), schwache Monotonie (Def. 3.11) und Grenzbedingung (Def. 3.15).

Die wichtigsten t- und s-Normen sind die min-/max-Norm, das algebraische Produkt/die algebraische Summe, die begrenzte Differenz/die begrenzte Summe, der drastische Schnitt/ die drastische Vereinigung und der Yager-Schnitt bzw. die Yager-Vereinigung, die in der Literatur bereits ausführlich diskutiert sind und daher an dieser Stelle nicht näher betrachtet werden (z.B. [73]).

Aufgrund der Tatsache, dass diese Normen nicht gewichtet, i.d.R. nicht stetig, nicht parametrisiert und nur schwach monoton sind, und daher nur sehr grobe Abstufungen der Relevanzen ermöglichen, sind sie für das gewichtete Retrieval nur bedingt geeignet. Insbesondere die häufig verwendeten min- bzw. max-Normen, die lediglich die höchste bzw. niedrigste Relevanz berücksichtigen, vernachlässigen die mittleren Zugehörigkeitsgrade vollständig: Ein Dokument, welches einen von fünf eingegebenen Suchbegriffen erhält, bekommt dieselbe Relevanz wie ein Dokument, welches vier der fünf Begriffe enthält.

3.3.2.2 TGQ, Generalized Quantifiers

Die „Theory of Generalized Quantifiers" (auch TGQ oder LGQ, „Logic with Generalized Quantifiers") wurde Anfang der achtziger Jahre vor einem linguistischen Hintergrund entwickelt ([3], [67]). Aufbauend auf dem bekannten All-Quantor ($\forall$) und dem Existenz-Quantor ($\exists$) werden verallgemeinerte Quantoren wie „most" und „some" entwickelt. Ihnen liegt ein relationales Modell zugrunde, d.h. ein Quantor Q wird als Relation zwischen zwei Mengen aufgefasst:

$$\begin{aligned} \text{all } X\ Y &\Leftrightarrow X \subseteq Y \\ \text{most } X\ Y &\Leftrightarrow |X \cap Y| > |X \backslash Y| \end{aligned}$$

$$\begin{aligned} \text{some } X\ Y &\Leftrightarrow X \cap Y \neq \emptyset \\ \text{no } X\ Y &\Leftrightarrow X \cap Y = \emptyset \\ \text{atleast-k-percent } X\ Y &\Leftrightarrow |X \cap Y| > \frac{k}{100} * |Y| \\ &\ldots \end{aligned}$$

Beispielsweise ist „most X Y" - im Sinne von „die meisten X sind Y" demnach genau dann erfüllt, wenn die Kardinalität der Schnittmenge von X und Y größer ist als die Kardinalität von X *ohne* Y, also so, wie es der natürliche Sprachgebrauch impliziert.

Ausgehend von dieser relationalen Sichtweise werden die verallgemeinerten Quantoren auf die üblichen Eigenschaften von Relationen hin untersucht, also z.B. Transitivität, Reflexivität, Symmetrie, Monotonie etc:

$$\begin{aligned} \text{Transitivität} &\Leftrightarrow (Q\ X\ Y\ \wedge\ Q\ Y\ Z) \rightarrow (Q\ X\ Z) \\ \text{Reflexivität} &\Leftrightarrow (Q\ X\ X) \\ \text{Symmetrie} &\Leftrightarrow (Q\ X\ Y) \rightarrow (Q\ Y\ X) \\ &\ldots \end{aligned}$$

So ist z.B. der Quantor „most" reflexiv, aber nicht symmetrisch, wohingegen der Quantor „some" beide Eigenschaften erfüllt.

Um die TGQ auf das gewichtete Retrieval anzuwenden, wäre X als die Menge der *vom Benutzer eingegebenen Stichworte*, und Y als die Menge der *in einem Dokument enthaltenen Stichworte* zu interpretieren. Dadurch ließen sich z.B. Anfragen wie „at least 3 of the keywords a,b,c,d,e" formulieren, was eine Verbesserung gegenüber dem klassischen, booleschen Retrieval, welches nur die Quantoren „alle" und „mindestens eines" zur Verfügung stellt, bedeutet.

Da die TGQ jedoch auch nur auf booleschen Mengen arbeitet, wäre eine Gewichtung der Suchbegriffe bzw. eine graduelle Relevanz in Abhängigkeit des Auftauchens eines Stichwortes im Abstract oder im Titel nicht möglich. Im weiteren werden daher Versuche, die „Generalized Quantifiers" auf Fuzzy-Mengen auszuweiten, vorgestellt.

3.3.2.3 FLQ (Fuzzy Linguistic Quantifiers), Σ-/FE-Count

Aufbauend auf der „Theory of Generalized Quantifiers" kann eine Klasse von „Fuzzy Linguistic Quantifiers" definiert werden (vgl. z.B. Zadeh [72]). Durch die Definition eines adäquaten Kardinalitätsmaßes wurde versucht, das Konzept der TGQ nahezu direkt zu übernehmen, dabei aber um Fuzzy-Mengen bzw. Fuzzy-Relationen zu erweitern und so die mangelnde Flexibilität der TGQ aufzuheben:

$$\Sigma Count(F) := \sum_{x \in F} \mu_F(x)$$

Basierend auf dem $\Sigma Count$ wird der sog. „relative count" (auch „relative cardinality") definiert, die den *Anteil* der Elemente x der Fuzzy-Menge X, die auch in der Fuzzy-Menge Y enthalten sind, beschreibt:

$$\Sigma Count(Y|X) := \frac{\Sigma Count(Y \cap X)}{\Sigma Count(X)} \quad \in [0..1]$$

Dabei können verschiedene Operatoren (insbesondere die in Kap. 3.3.2.1 vorgestellten t-Normen) zur Schnittmengenbildung ($Y \cap X$) dienen.

Ein Fuzzy-Quantor ist dann definiert als eine Funktion in Abhängigkeit des relative count:

$$\mu_{agg} := \mu_{agg}(\Sigma Count(Y|X))$$

So ist bei einer Anwendung der FLQ auf das gewichtete Retrieval analog zur TGQ (Kap. 3.3.2.2) X als die Menge der *wichtigen, vom Benutzer eingegebenen Stichworte*, und Y als die Menge der *in einem Dokument enthaltenen Stichworte* zu interpretieren. Im Gegensatz zur TGQ ist hier der Zugehörigkeitsgrad eines Stichwortes zu der Menge X der für den Benutzer *wichtigen* Dokumente eine *graduelle* Gewichtung des Suchbegriffes. Die ebenfalls graduellen Zugehörigkeitsgrade der Stichworte zu den im Dokument *enthaltenen* Stichworten Y drücken die Relevanz eines Suchterms in Bezug auf ein Dokument aus. Das gelieferte Ergebnis μ_{agg} ist ebenfalls graduell und drückt die Relevanz eines Dokumentes aus. Dazu ein Beispiel: Gegeben sei der Aggregations-Operator „many" mit $many(Y|X) := S(\Sigma Count(Y|X), 0.3, 0.5, 0.7)$, dargestellt in Abb. 3.4, wobei S Zadehs S-Funktion ist:

$$S(x,a,b,c) = \begin{cases} 0 & für \quad x \leq a \\ 2((x-a)/(c-a))^2 & für \quad a \leq x \leq b \\ 1 - 2((x-c)/(c-a))^2 & für \quad b \leq x \leq c \\ 1 & für \quad x \geq c \end{cases}$$

Gegeben sei eine Anfrage nach „fuzzy", „information" und „retrieval" mit den Gewichten $w_{fuzzy} = 1.0$, $w_{information} = 0.5$ und $w_{retrieval} = 0.8$. Ein Dokument mit dem Titel „Fuzzy logic", bei dem der Begriff „retrieval" nur im Abstract auftaucht, habe die Zugehörigkeitsgrade $\mu_{fuzzy} = 1.0$, $\mu_{information} = 0.0$ und $\mu_{retrieval} = 0.4$. Als Schnittmengenoperator diene das algebraische Produkt. Die Gesamt-Relevanz des Dokumentes in Bezug auf die Anfrage berechnet sich dann als

$$\begin{aligned} \mu_{agg} &= S(\Sigma Count(Y|X), 0.3, 0.5, 0.7) \\ &= S((1.0 * 1.0 + 0.5 * 0.0 + 0.8 * 0.4)/(1.0 + 0.5 + 0.8), 0.3, 0.5, 0.7) \\ &\simeq S(0.57, 0.3, 0.5, 0.7) \\ &\simeq 0.80 \end{aligned}$$

Der FG-Count-Ansatz, ein dem $\Sigma Count$ recht ähnlicher, alternativer Ansatz wird ebenfalls von Zadeh in [72] vorgeschlagen. Als Kardinalität einer Fuzzy-Menge wird hier nicht die Summe der Zugehörigkeitsgrade, also ein *numerisches Skalar* verwendet, sondern seinerseits eine *Fuzzy-Zahl*. Dazu werden zunächst die Hilfsgrößen FG- und FL-Count eingeführt:

$$\begin{aligned} \mu_{FGCount(F)}(i) &:= sup_\alpha\{\alpha : |(F)_\alpha| \geq i\} \\ \mu_{FLCount(F)}(i) &:= sup_\alpha\{\alpha : |(F)_\alpha| \geq n - i\} \end{aligned}$$

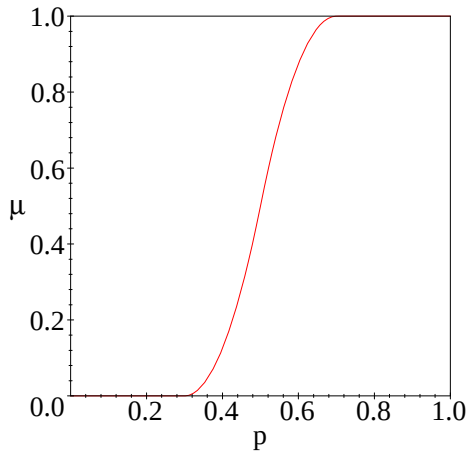


Abbildung 3.4: Der durch den $\Sigma Count$ definierte Fuzzy-Quantor „many". Zugehörigkeitsgrad in Abhängigkeit des Anteils p von X an Y.

Dabei ist $(F)_\alpha$ der schwache α-Schnitt (vgl. Def. 3.2) und $|.|$ die Kardinalität der (booleschen) Menge. Der FG-Count ist also eine Fuzzy-Zahl, die den Grad ausdrückt, zu dem die Fuzzy-Menge F *mindestens* i Elemente hat, wohingegen der FL-Count den Grad ausdrückt, zu dem F *höchstens* i Elemente hat. Der FE-Count ist dann als Schnitt dieser Fuzzy-Zahlen definiert und drückt demnach den Grad aus, zu dem die Fuzzy-Menge F *genau* i Elemente hat:

$$\mu_{FECount(F)}(i) \quad := \quad \mu_{FGCount(F)}(i) \wedge \mu_{FLCount(F)}(i)$$

Als Konjunktion ($\wedge$) schlägt Zadeh hier die in Kapitel 3.3.2.1 erwähnte min-Norm vor. Der FG-Count der Fuzzy-Menge $F = fuzzy/1.0 + information/0.5 + retrieval/0.8$ berechnet sich z.B. wie folgt:

$$\begin{aligned} FGCount(F) &= 0/1.0 + 1/1.0 + 2/0.8 + 3/0.5 + 4/0.0 \\ FLCount(F) &= 0/0.5 + 1/0.8 + 2/1.0 + 3/1.0 + 4/1.0 \\ FECount(F) &= 0/0.5 + 1/0.8 + 2/0.8 + 3/0.5 + 4/0.0 \end{aligned}$$

Abbildung 3.5 stellt diese Fuzzy-Zahlen zusammen mit dem $\Sigma Count$, der - da es keine Fuzzy-Zahl ist - nur als Peak zu erkennen ist, graphisch dar.

Diese naheliegenden und zunächst vielversprechenden Ansätze, die TGQ auf Fuzzy-Mengen zu erweitern, scheitern jedoch u.a. an den in Satz 3.1 festgestellten Inkonsistenzen: Es ist keine einheitliche Definition eines neutralen Elementes bzw. keine Definition der Grenzbedingungen (nach Def. 3.15) möglich, und die Parametrisierungs-Eigenschaft (Def. 3.16) kann für monotone Operatoren (wie „möglichst alle" oder „möglichst viele"), wie sie beim ge-

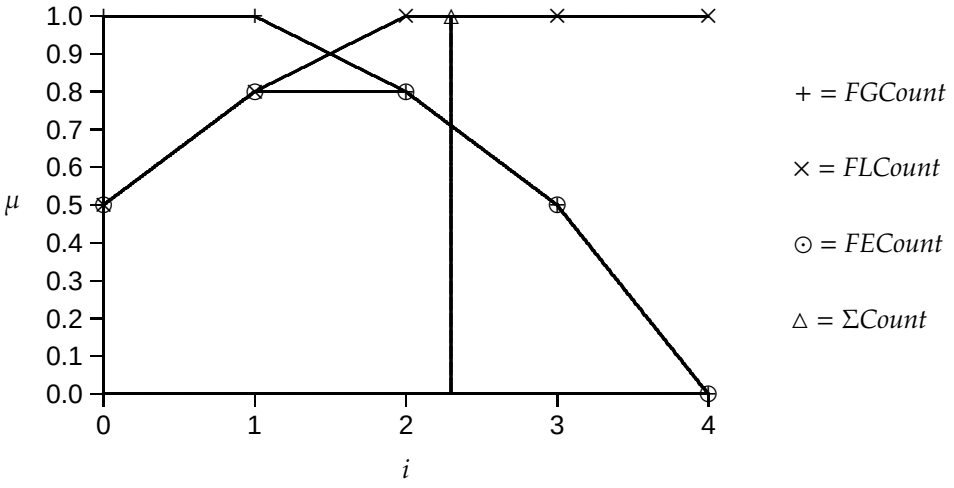


Abbildung 3.5: $FG-$, $FL-$, $FE-$ und $\Sigma Count$ der Fuzzy-Menge $F = fuzzy/1.0 + information/0.5 + retrieval/0.8$

wichteten Retrieval benötigt werden, nicht eingehalten werden (Beweis siehe Anhang B.2). Für das gewichtete Retrieval sind diese Kardinalitätsansätze daher nicht geeignet.

3.3.2.4 OWA

OWA-Operatoren („ordered weighted averaging", [70]) stellen eine Klasse von Fuzzy-Quantoren dar, bei denen die Gewichte intrinsisch mit dem Operator verknüpft sind, also nicht explizit vom Benutzer angegeben werden: Die Klasse der Operatoren ist definiert durch das Skalarprodukt des Vektors der normierten Gewichte und des Vektors aus den - *absteigend sortierten* - Zugehörigkeitsgraden:

$$OWA_W(\mu_1, ..., \mu_n) := \sum_{i=1}^{n} w_i * \mu_i$$

$$mit \quad \sum_{i=1}^{n} w_i = 1 \quad \wedge \quad \forall_i w_i \in [0..1]$$

$$und \quad \mu_i \leq \mu_j \quad für \quad i > j$$

Die verschiedenen Ausprägungen dieser Klasse von Operatoren sind definiert durch den festgelegten Gewichtsvektor; beispielsweise realisiert der Vektor $W^* = [1\ 0\ ...\ 0]^T$ die max-Norm, $W_* = [0\ ...\ 0\ 1]^T$ die min-Norm, $W_{Med} = [0\ ...\ 0\ 1\ 0\ ...\ 0]$ den Median und $W_{Ave} = [\frac{1}{n}\ ...\ \frac{1}{n}]^T$ das arithmetische Mittel.

Das bereits in Kapitel 3.3 erwähnte „Orness"-Maß ist wie folgt definiert:

$$orness(W) = \frac{1}{n-1} \sum_{i=1}^{n} ((n-i) * w_i)$$

Es drückt den Grad aus, wie ähnlich ein Operator dem W^*-Operator, also dem ODER-Operator ist.

OWA-Operatoren sind kommutativ, nicht generell assoziativ, schwach monoton, stetig und erfüllen die Grenzbedingungen, wie man leicht herleiten kann. Einige sind darüber hinaus streng monoton.

Der Hauptnachteil für das gewichtete Retrieval besteht darin, dass durch die Sortierung der μ_i keine substantielle Korrelation zwischen Gewicht und Suchbegriff vorhanden ist. Werden beispielsweise zwei Suchbegriffe vom Benutzer eingegeben, und die Gewichte liegen bei 0.8 und 0.2, dann ist es irrelevant, *welcher* der beiden Suchbegriffe in einem Dokument enthalten ist. Das Dokument bekommt in beiden Fällen dieselbe Relevanz. Die Gewichte haben also weniger die Funktion, die „Wichtigkeit" eines Suchbegriffes festzulegen, vielmehr sind sie Parameter des Operators, welche die Orness beeinflussen.

Aus diesem Grund (und aus dem Mangel einiger dieser Operatoren an strenger Monotonie, der die in Kapitel 3.3 diskutierten Nachteile nach sich zieht) sind OWA-Operatoren für das gewichtete Retrieval nur von geringem Nutzen.

3.3.2.5 p-Norm und WAO$_\lambda$-Operator

An dieser Stelle wird ein gewichteter Fuzzy-Quantor untersucht, der alle in Kapitel 3.3 geforderten Eigenschaften erfüllt. Er wird im weiteren als WAO_λ („weighted averaging operator") bezeichnet, der durch einen Parameter λ parametrisiert ist:

Definition 3.17: WAO_λ

Der Fuzzy-Quantor WAO_λ ist definiert als:

$$WAO_\lambda(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) := \left(\frac{\sum (w_i * \mu_i)^\lambda}{\sum w_i^\lambda} \right)^{\frac{1}{\lambda}} \text{ mit } \lambda > 0$$

In der Literatur (z.B. [61]) wird dieses Maß oft als *p-Norm* bezeichnet, jedoch ist der Wertebereich für den Parameter dort eingeschränkt auf $p \geq 1$, wodurch unter anderem die wichtige Parametrisierungs-Eigenschaft verloren geht. Stattdessen konzentriert sich die Betrachtung dort auf eine andere „Dualität", die nach De Morgan erreicht wird; statt eines einzigen Operators werden die beiden Operatoren OR_p und AND_p definiert:

$$OR_p = \left(\frac{\sum (w_i * \mu_i)^p}{\sum w_i^p} \right)^{\frac{1}{p}}$$

$$AND_p \quad = \quad 1 - \left(\frac{\sum (w_i * (1 - \mu_i))^p}{\sum w_i^p} \right)^{\frac{1}{p}}$$

Wenn man den AND_p-Operator als Dual des OR_p-Operators interpretiert, so kann durch geeignete Wahl von p die Intention hinter der Parametrisierungs-Eigenschaft, zwischen den beiden in Kapitel 3.3 genannten Aspekten (*möglichst viele der eingegebenen Suchbegriffe* und *möglichst die wichtigsten Suchbegriffe* zu berücksichtigen) gewichten zu können, zwar erfüllt werden, jedoch muss p dazu in Abhängigkeit von der Dimension und den Gewichten bestimmt werden, und es kann nicht - wie in der Parametrisierungs-Bedingung gefordert - ein globales p für alle Dimensionen und Gewichte festgelegt werden (Beweis siehe Anhang B.3). Weiterhin genügt nur der OR_p-Operator auch allen anderen geforderten Eigenschaften - der AND_p-Operator hingegen ist z.B. nicht k-monoton, wie in Anhang B.5 gezeigt wird. Darüber hinaus beinhaltet die p-Norm interessante Spezialfälle nicht, die im Folgenden noch diskutiert werden.

Im Gegensatz zur p-Norm erfüllt der WAO-Operator alle Anforderungen nach Kommutativität (Def. 3.9), strenger ∃-Monotonie bezüglich Zugehörigkeitsgraden und Gewichten (Def. 3.11, 3.12), K-Monotonie (Def. 3.13), Stetigkeit bezüglich aller Argumente (Def. 3.14), Grenzbedingungen (Def. 3.15.1-.3) und die Parametrisierungs-Bedingung (Def. 3.16, vgl. Abb. 3.6). p-Norm und WAO-Operator sind nicht generell assoziativ (Beweis siehe Anhang B.6) was, wie bereits erläutert, der intuitiven Vorstellung entspricht, dass ein Term „Agg(a,b,c, Agg(x,y,z))" ganz offensichtlich mehr Wert auf die Sub-Terme a,b,c legt als auf die lediglich verschachtelt vorkommenden x,y,z (wohingegen bei einem assoziativen Operator der Term äquivalent wäre zu „Agg(a,b,c,x,y,z)", bei dem alle a,b,c,x,y,z gleichwertig wären).

Die Beweise der Kommutativität, strenger ∃-Monotonie bezüglich der Zugehörigkeitsgrade, Stetigkeit in allen Argumenten und der Grenzbedingungen (Def. 3.15.1-.3) sind trivial und werden hier nicht aufgeführt. Für die Beweise der Grenzbedingung nach Def. 3.15.8, der strengen ∃-Monotonie bezüglich der Gewichte, der K-Monotonie und der Parametrisierung sei auf Anhang B.7-B.10 verwiesen. Damit erfüllt der WAO_λ-Operator alle geforderten Eigenschaften.

Anhang C zeigt Graphen des WAO-Operators für $n = 2$ in Abhängigkeit von $\langle w_1, \mu_1 \rangle$ für $w_2 = 1$ und $\mu_2 = 1$ bzw. 0 und in Abhängigkeit von w_1 und w_2, jeweils für verschiedene Werte von λ. Aufgrund der Vielzahl an Parametern (λ, w_1, μ_1, w_2, μ_2, ..., w_n, μ_n) werden einige Eigenschaften nicht deutlich, was jedoch im Folgenden anhand von Spezialfällen betrachtet wird.

Spezialfall 1: Arithmetisches Mittel

Für $\lambda = 1$ und boolesche Gewichte ($w_i = 1$) reduziert sich der Operator auf das *arithmetische Mittel* der Zugehörigkeitsgrade μ_i:

$$WAO_1(\langle 1, \mu_1 \rangle, ..., \langle 1, \mu_N \rangle) = \left(\frac{\sum_{i=1}^{N} (1 * \mu_i)^1}{\sum_{i=1}^{N} 1^1} \right)^{\frac{1}{1}} = \frac{1}{N} \sum_{i=1}^{N} \mu_i$$

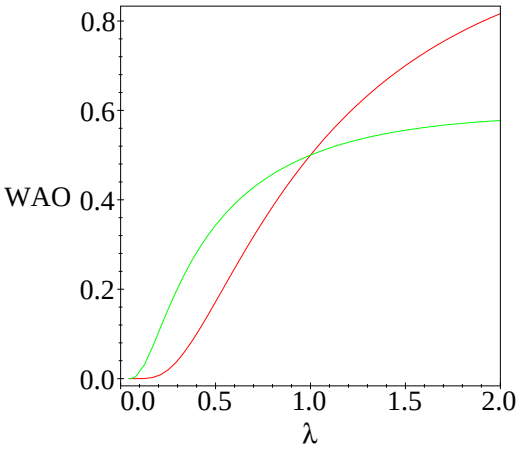


Abbildung 3.6: WAO_λ-Operator in Abhängigkeit von λ für $(\langle 1.0, 1.0\rangle, \langle 0.5, 0.0\rangle, \langle 0.5, 0.0\rangle)$ (rot) und $(\langle 1.0, 0.0\rangle, \langle 0.5, 1.0\rangle, \langle 0.5, 1.0\rangle)$ (grün). Hier ist deutlich die Parametrisierungs-Eigenschaft zu erkennen, bei der - in Abhängigkeit von λ - entweder ein Dokument mit einem *wichtigen* Stichwort oder aber ein Dokument mit *mehreren*, aber dafür unwichtigeren Stichworten bevorzugt wird (vgl. Abb. 3.3).

In diesem Fall liegt eine lineare Abhängigkeit des Operators von den μ_i vor; Monotonie, Kommutativität und Grenzbedingungen sind hier besonders offensichtlich.

Spezialfall 2: Gewichteter Quotient

Sind sowohl die Gewichte w_i, als auch die Zugehörigkeitsgrade μ_i boolesch, so erhält man den gewichteten Quotienten

$$WAO_\lambda(\langle 1, \mu_1\rangle, ..., \langle 1, \mu_N\rangle) = \left(\frac{\sum_{i=1}^{N}(1 * \mu_i)^\lambda}{\sum_{i=1}^{N} 1^\lambda}\right)^{\frac{1}{\lambda}} = \left(\frac{M}{N}\right)^{\frac{1}{\lambda}}$$

wobei N die Anzahl an Zugehörigkeitsgraden und M die Anzahl an *wahren* Zugehörigkeitsgraden ist. Für das Beispiel des Dokument-Retrievals bedeutet dies, dass die Relevanz eines Dokumentes nur noch von dem Anteil der im Dokument enthaltenen Stichworte an der Gesamtzahl der vom Benutzer angegebenen Stichworten abhängig ist. Abbildung 3.7 stellt diesen Spezialfall für verschiedene M (in Prozent von N) und für unterschiedliche λ dar. Hier ist sehr gut der Einfluss von λ zu erkennen: λ wirkt als eine Art „Toleranzfaktor", der festlegt, wie stark die Gesamt-Relevanz eines Dokumentes ansteigen soll, wenn nur einige Stichworte enthalten sind. Für kleine Werte ($\lambda < 1$) steigt die Relevanz nur langsam an,

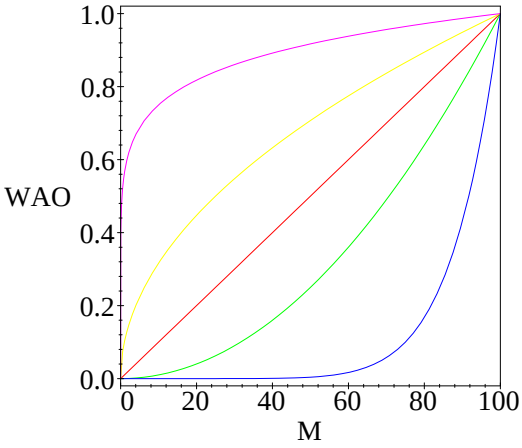


Abbildung 3.7: WAO_λ-Operator mit $w_i = 1$ und booleschen μ_i in Abhängigkeit des Anteils M an im Dokument enthaltenen Stichworten (für $\lambda = 8, 2, 1, 1/2$ und $1/8$). Während der „einige"-Operator schnell ansteigt, ist bei dem „möglichst alle"-Quantor ein höherer Anteil an Stichworten nötig, um eine hohe Relevanz zu erhalten.

während für große Werte ($\lambda > 1$) bereits wenige Stichworte für eine relativ hohe Relevanz ausreichen.

Spezialfall 3: Maximums-Norm

Bei booleschen Gewichten ($w_i = 1$) ist insbesondere der Grenzwert $\lambda \to \infty$ interessant, der die bekannte Maximums-Norm (vgl. Kapitel 3.3.2.1), also eine Interpretation des ODER-Operators liefert:

$$WAO_\infty(\langle 1, \mu_1\rangle, ..., \langle 1, \mu_N\rangle) := \lim_{\lambda\to\infty}\left(\frac{\sum_{i=1}^{N}(1 * \mu_i)^\lambda}{\sum_{i=1}^{N} 1^\lambda}\right)^{\frac{1}{\lambda}} = \frac{\lim(\sum_{i=1}^{N}\mu_i^\lambda)^{\frac{1}{\lambda}}}{\lim N^{\frac{1}{\lambda}}} = \max_i \mu_i$$

Spezialfall 4: Geometrisches Mittel

Der Grenzwert $\lambda \to 0$ liefert - unabhängig davon, ob die Gewichte boolesch sind - das bekannte geometrische Mittel, also einen UND-Operator:

$$WAO_0(\langle w_1, \mu_1\rangle, ..., \langle w_N, \mu_N\rangle) := \lim_{\lambda\to 0}\left(\frac{\sum(w_i * \mu_i)^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} = \frac{(\prod w_i * \mu_i)^{\frac{1}{N}}}{(\prod w_i)^{\frac{1}{N}}} = \left(\prod_{i=1}^{N}\mu_i\right)^{\frac{1}{N}}$$

3.4 Anfrageoptimierungen

Bei klassischen, booleschen Informationssystemen bzw. relationalen und Volltextdatenbanken sind Anfrage-Optimierungen bereits weitgehend erforscht.

Das Ziel derartiger Anfrageoptimierungen ist es, eine Anfrage q in eine *äquivalente* Anfrage q' umzuformen, die die gleiche Ergebnismenge liefert, jedoch effizienter abzuarbeiten ist, und damit für eine höhere Performance des Systems sorgt.

Bei Fuzzy-Metasuchmaschinen ist das Ziel von Anfrageoptimierungen - erhöhte Performance und verkürzte Antwortzeiten - das gleiche, jedoch sind die Voraussetzungen und Teilziele andere:

- Fuzzy-Operatoren haben andere Eigenschaften als boolesche Operatoren, wie im letzten Unterkapitel diskutiert wurde (bspw. fehlende Assoziativität), weswegen andere Arten von Optimierungen möglich und nötig sind.
- Die der Metasuchmaschine zugrundeliegenden Datenbanken liegen i.d.R. nicht lokal vor. Es muss also auf eine bereits bestehende Indexierung zurückgegriffen werden, und es können keinen eigenen, inversen Index-Dateien aufgebaut werden. Als zeitkritischer Faktor ist wegen des Netzwerk-Flaschenhalses die zu übertragende Datenmenge und damit die Anzahl der Zugriffe auf die Datenbasis ausschlaggebend.
- Das auf iterativer Berechnung einer sortierten Ergebnisliste basierende Streaming-Prinzip erfordert eine besonderen Beachtung: Es gilt nicht vorrangig, das komplette *Gesamtergebnis* möglichst schnell zu liefern, sondern das Hauptziel ist es, möglichst schnell *Teilergebnisse* sukzessive in den Ergebnisstream leiten zu können.

Die wichtigsten Optimierungen, die sich für eine Fuzzy-Metasuchmaschine ergeben, sind *Minimierung von Teilergebnismengen*, *Delegation von booleschen Teilanfragen* und *Komposition von Aggregations-Operatoren*.

3.4.1 Ausgangslage

Im weiteren wird von folgender Anfragestruktur ausgegangen (Notation in EBNF):

$$
\begin{array}{lll}
query = & Atomic_{spec} & | \\
& AND([query]) & | \\
& OR([query]) & | \\
& NOT(query) & | \\
& Agg_{spec}([query]) & | \\
& Filter_{spec}(query) &
\end{array}
$$

Eine Anfrage kann demnach *atomar* sein, also z.B. eine Stichwortsuche repräsentieren. Sie kann einer der üblichen booleschen Fälle, also eine *Konjunktion*, eine *Disjunktion* oder eine

Negation darstellen. Die *Aggregation* ist die Verallgemeinerung dieser booleschen Operatoren auf die Fuzzy-Logik bestehend aus der Spezifikation des Quantors (z.B. „einige") und einer Liste von Unteranfragen. (Auf die Spezifikation der Gewichte wird an dieser Stelle der Einfachheit halber verzichtet, da sie für die Anfrageoptimierung nicht relevant sind.) Eine *Filter*-Anfrage ist ebenfalls eine Fuzzy-Anfrage, die aus einer Spezifikation des Filters (z.B. „Datum möglichst neu") und einer Unteranfrage besteht. (Eine Anfrage „Datum möglichst neu" kann also nicht alleine ausgeführt werden kann, da sie lediglich die Zugehörigkeitsgrade der Elemente einer bereits vorhandenen Ergebnismenge nachträglich anhand eines Attributs beeinflusst).
Eine (Teil-)Anfrage, die rekursiv lediglich aus Konjunktionen, Disjunktionen, Negationen und atomaren Teilanfragen besteht, und keine Aggregations- und Filter-Operatoren enthält, und somit direkt von dem Wrapper bzw. der Datenbank ausgeführt werden kann, wird im Folgenden kurz als *boolesch* bezeichnet, andernfalls als *fuzzy*.

3.4.2 Minimierung von Teilergebnismengen

Wie eingangs erwähnt zieht die Metasuchmaschinen-Architektur den Flaschenhals des Netzwerks nach sich, über welchen auf die zumeist nicht lokal vorliegenden Datenbanken zugegriffen wird. Um eine kurze Antwortzeit zu gewährleisten, gilt es also, nach Möglichkeit Teilanfragen so zu konstruieren, dass die von den Datenbanken gelieferten Ergebnisse bereits einen geringen Umfang haben.

Ein Beispiel verdeutlicht das: Abbildung 3.8 zeigt den Anfragebaum einer einfachen Anfrage nach dem Stichwort „electronics" mit dem fuzzy Filter-Operator „möglichst neu". Der maximale boolesche Teilbaum, der von einer Datenbank ausgeführt werden kann, ist hier die atomare Anfrage nach dem Stichwort und ist in der Grafik markiert.

Abbildung 3.9 zeigt eine äquivalente Anfrage, die jedoch - trotz höherer Komplexität - performanter bearbeitet werden kann: Die boolesche Hülle (vgl. Def. 3.3) des Filter-Operators wird hier zunächst konjunktiv mit der Anfrage nach dem Stichwort verknüpft, und anschließend wird die Filter-Operation durchgeführt. Die Konsequenz ist, dass ein größerer Teilbaum der Anfrage boolesch ist und direkt von der Datenbank verarbeitet werden kann, was die Teilergebnismenge, die von der Datenbank an den Fuzzy-Query-Interpreter übertragen werden muss, bereits frühzeitig einschränkt, und somit die Netzlast als begrenzenden Faktor gering hält.

Generell kann dieser Aspekt der Anfrageoptimierung wie folgt formalisiert werden:

Theorem 3.3: Anfrageoptimierung: Minimierung von Teilergebnismengen

> Gegeben sei eine Anfrage $q = Filter_{spec}(q_s)$ bestehend aus einer Filter-Anfrage und einer Teilanfrage q_s. Weiterhin sei $envelope(spec)$ die boolesche Hülle der Filter-Anfrage. Dann kann q in die äquivalente, optimierte Anfrage q' transformiert werden:

$q' = Filter_{spec}(q_{DB})$ mit $q_{DB} = AND(envelope(spec), q_s)$

Ist q_s boolesch, so ist auch q_{DB} boolesch und kann direkt von der Datenbank ausgeführt werden.

Zu beweisen ist, dass die optimierte Anfrage äquivalent zu der Originalanfrage ist, also die gleichen Relevanzen liefert:

Beweis:

z.z.: $\mu_{filter_{spec}} * \mu_s = \mu_{filter_{spec}} * t(\mu_{envelope(spec)}, \mu_s)$

Fallunterscheidung:

(a) Sei $\mu_{filter_{spec}} = 0$. Dann folgt unmittelbar:

$$\begin{aligned} &\mu_{filter_{spec}} * \mu_s \\ &= 0 \\ &= \mu_{filter_{spec}} * t(\mu_{envelope(spec)}, \mu_s) \end{aligned}$$

(b) Sei $\mu_{filter_{spec}} > 0$. Dann gilt:

$$\begin{aligned} &\mu_{filter_{spec}} * \mu_s \\ &= \mu_{filter_{spec}} * t(1, \mu_s) && \text{(Def. Grenzbedingung 3.15)} \\ &= \mu_{filter_{spec}} * t(\mu_{envelope(spec)}, \mu_s) && \text{(Def. Boolesche Hülle 3.3)} \end{aligned}$$

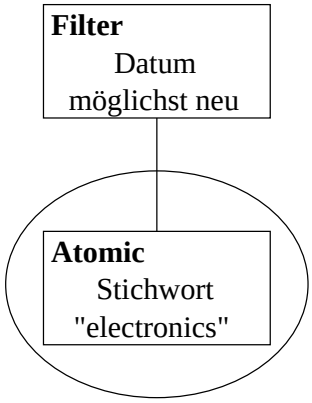


Abbildung 3.8: Nicht Optimierte Anfrage mit einem fuzzy Filter-Operator. Die von der Datenbank ausführbare Teilanfrage ist durch eine Ellipse markiert und liefert eine noch sehr große Ergebnismenge, die einer Nachbearbeitung unterzogen werden muss.

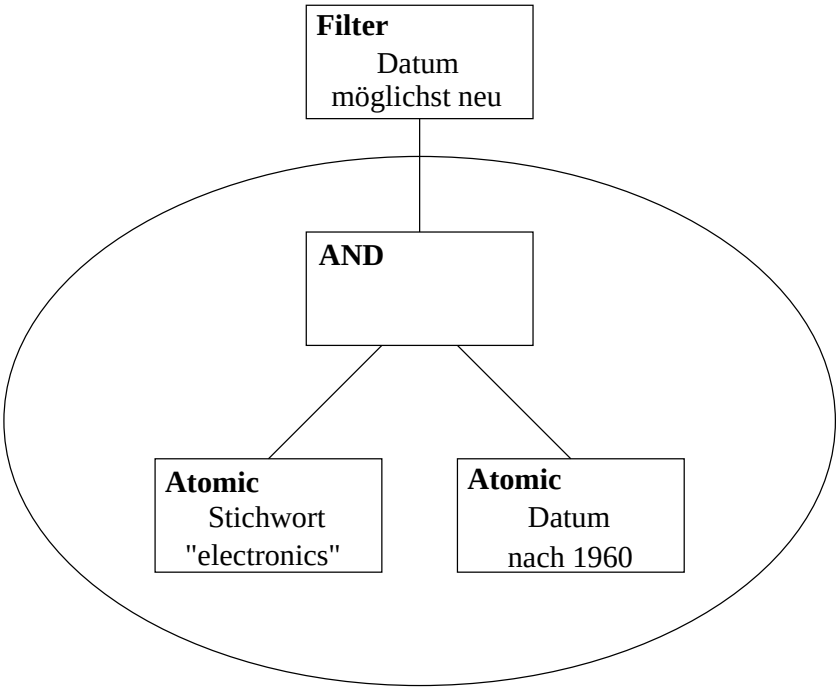


Abbildung 3.9: Optimierte Anfrage mit einem fuzzy Filter-Operator. Die von der Datenbank ausführbare Teilanfrage ist durch eine Ellipse markiert und liefert im Gegensatz zu der nicht optimierten Anfrage (Abb. 3.8) eine besser handhabbare Ergebnismenge.

3.4.3 Delegation von booleschen Teilanfragen

Basieren die in die Metasuchmaschine eingebundenen Datenbanken auf der *booleschen Logik* (und erlauben selbst kein gewichtetes Retrieval), so ist es sinnvoll, möglichst komplexe Teile der Anfragen von den Datenbank ausführen zu lassen, da boolesche Anfragen performanter abgearbeitet werden können, und nur noch wenig Arbeit für die aufwändige Berechnung der Fuzzy-Zugehörigkeitsgrade vom Fuzzy-Query-Interpreter geleistet werden muss.

Dies kann, wie in dem folgende Beispiel, im Widerspruch zu dem Ziel der „Minimierung der Teilergebnismengen" stehen, jedoch sind die Performance-Gewinne durch die Delegation deutlich größer als die Verluste, die sich durch die Vergrößerung der Teilergebnisse ergeben. Der Grund ist, dass oft nur die ersten Datensätze eines Ergebnisses angefordert werden, und der Netzwerk-Flaschenhals dann weniger stark ins Gewicht fällt.

Abbildung 3.10 zeigt einen Anfragebaum bestehend aus einer Disjunktion und einer Aggregation. Aufgrund der Tatsache, dass der vom Fuzzy-Query-Interpreter evaluierte Aggregations-Operator eine Fuzzy-Menge als Ergebnis liefert, muss auch die darüberliegende Disjunktion mittels der Fuzzy-Logik interpretiert werden. Abbildung 3.11 zeigt eine äquivalente, jedoch optimierte Anfrage. Da hier die Disjunktion nur boolesche Teilanfragen verknüpft, und daher selbst boolesch ist, kann sie direkt von der Datenbank ausgeführt werden. Der Fuzzy-Query-Interpreter muss daher lediglich noch eine einzige, aufwändige Fuzzy-Aggregation durchführen.

Diese Optimierung wird wie folgt formalisiert:

Theorem 3.4: Anfrageoptimierung: Delegation von booleschen Teilanfragen

> Gegeben sei eine disjunktive Anfrage $q = OR(q_A, q_{b_1}, ..., q_{b_B})$ mit einer Aggregations-Teilanfrage $q_A = Agg_{spec}(q_{a_1}, ..., q_{a_A})$ aus rein booleschen Teilanfragen $q_{a_i} \quad (i = 1..A)$ und den booleschen Teilanfragen $q_{b_i} \quad (i = 1..B)$. Dann kann q in die äquivalente, optimierte Anfrage q' transformiert werden:
>
> $q' = Agg_{spec}(q'_{a_1}, ..., q'_{a_A})$ mit $q'_{a_i} = OR(q_{a_i}, q_{b_1}, ..., q_{b_B})$
>
> Dabei sind die q'_{a_i} rein boolesch und direkt von der Datenbank ausführbar.
>
> Die Optimierung ist analog auf Konjunktionen übertragbar. In diese Fall wird sogar die Ergebnismenge eingeschränkt, so dass sich auch die in Kapitel 3.3 diskutierten Vorteile auswirken.

Zu zeigen ist wieder die Äquivalenz der beiden Anfragen:

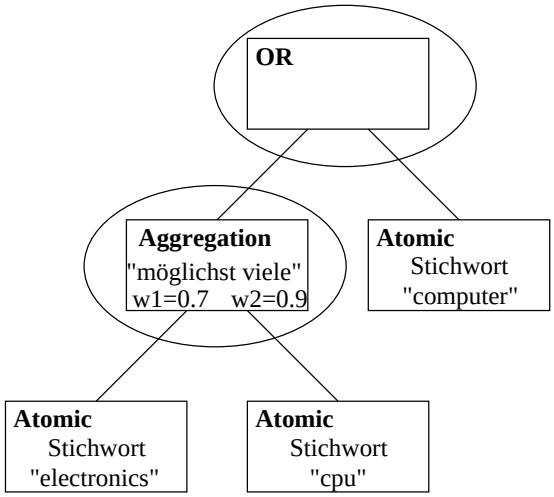


Abbildung 3.10: Nicht optimierte Anfrage mit einem fuzzy Aggregations-Operator und einem booleschen Operator. Aufgrund der Struktur der Anfrage müssen jedoch beide Operatoren vom Fuzzy-Query-Interpreter ausgeführt werden, was einen kalkulatorisch hohen Aufwand erfordert.

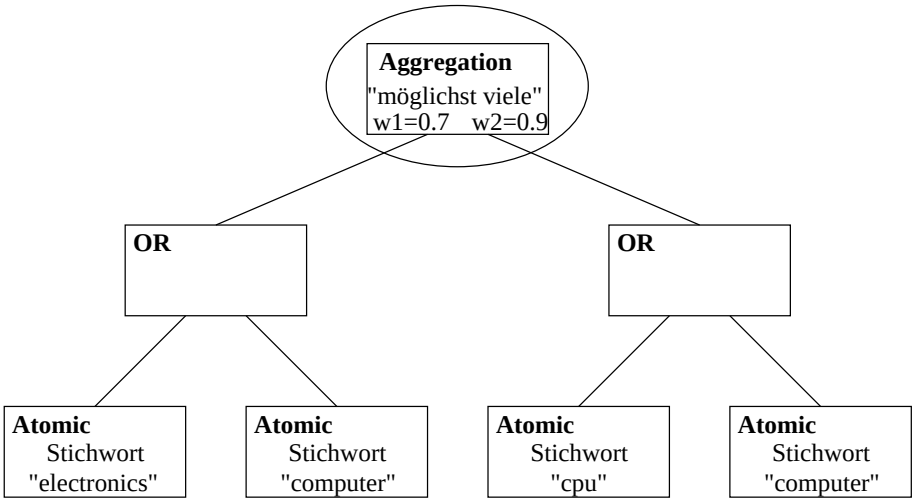


Abbildung 3.11: Optimierte Anfrage mit einem fuzzy Aggregations-Operator. Im Gegensatz zu der nicht optimierten Anfrage (Abb. 3.10) ist hier lediglich *eine* kalkulatorisch aufwändige Fuzzy-Aggregation durchzuführen.

Beweis:

Sei $\mathfrak{s}$ eine s- oder t-Norm, μ_{agg} ein Aggregations-Operator. Zu zeigen ist:

$$\mathfrak{s}(\mu_{agg}(\langle w_1, \mu_{a_1}\rangle, ..., \langle w_A, \mu_{a_A}\rangle), \mu_{b_1}, ..., \mu_{b_B}) = \mu_{agg}(\langle w_1, \mathfrak{s}(\mu_{a_1}, \mu_{b_1}, ..., \mu_{b_B})\rangle, ..., \langle w_A, \mathfrak{s}(\mu_{a_A}, \mu_{b_1}, ..., \mu_{b_B})\rangle)$$

Da q_{a_i} und q_{b_i} laut Voraussetzung boolesch sind, folgt dies direkt aus der partiellen Distributivität (Def. 3.2).

3.4.4 Komposition von Aggregations-Operatoren

Als letzte Möglichkeit der Optimierung bietet es sich an, verschiedene Aggregations-Operatoren zu einem zusammenzuführen. Die Idee der Komposition geht von atomaren Teilanfragen oder Aggregations-Unteranfragen aus und wird die Voraussetzung für die später vorgestellten Evaluationsstrategien schaffen. Diese Strategien skalieren im Wesentlichen mit der Anzahl von ineinander verschachtelten Aggregations-Operatoren (und damit der Tiefe des Anfragebaums), aber auch mit der Anzahl der im Aggregations-Operator eingebetteten Teilanfragen. Daher ist es wichtig, diese Größen wenn möglich gering zu halten. Die Komposition von Aggregationsoperatoren als Optimierung kann also nicht unabhängig von den in Kapitel 4 entwickelten Evaluationsstrategie betrachtet werden.

Da, wie gezeigt, die booleschen Operatoren Spezialfälle von Aggregations-Operatoren sind, können boolesche Konjunktionen und Disjunktionen ebenfalls durch Komposition optimiert werden. Es lassen sich also nahezu alle Klassen von Anfragen behandeln, ausgenommen sind lediglich Anfragen wie in Abb. 3.12, bei der die Ebenen des Anfragebaumes mit Aggregations-Operatoren durch eine Ebene mit einem Filter-Operator getrennt sind.

Abb. 3.14 zeigt ein Beispiel einer Optimierung der Anfrage in Abb. 3.13, wobei aus den Wertetabellen der ursprünglichen Aggregations-Operatoren die Wertetabelle für den kombinierten Aggregations-Operator abgeleitet wurde. Nachteil dieser Optimierung ist die in der Abbildung zu erkennende, erhöhte Anzahl an Unteranfragen des kombinierten Aggregations-Operators, was einen negativen Einfluss auf die Performance der Evaluation hat (auch dies wird in Kapitel 4 genauer untersucht). Aus diesem Grund sollten die anderen Optimierungsverfahren vorgezogen werden. Abbildung 3.15 zeigt eine ebenfalls optimierte, zu 3.13 äquivalente Anfrage, bei der zunächst die Optimierung 3.4 (Delegation) durchgeführt wurde. Diese Anfrage ist effizienter auszuführen, weil wie in der Grafik zu erkennen, nur drei anstelle von vier Teilanfragen ausgeführt werden müssen.

Die Komposition von Aggregations-Operatoren lässt sich wie folgt formalisieren:

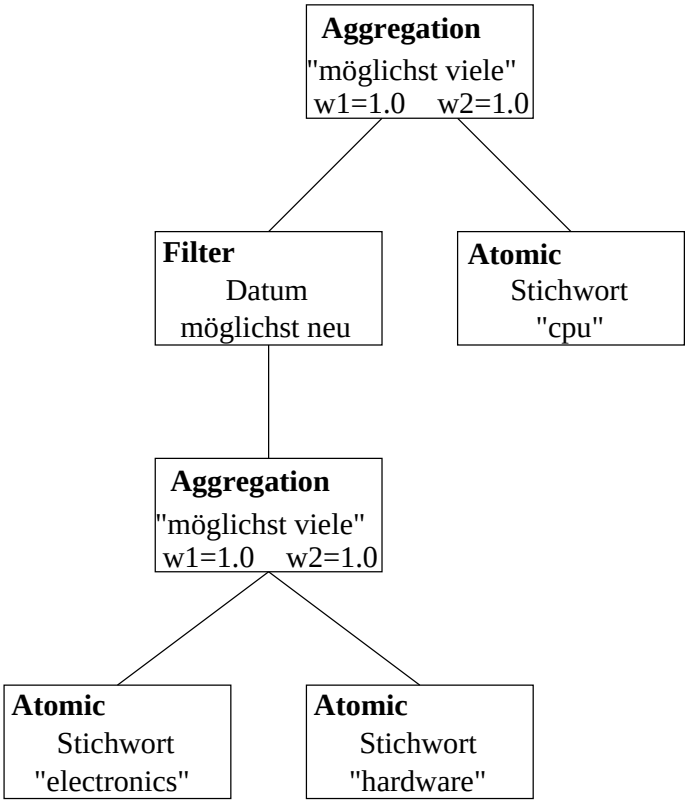


Abbildung 3.12: Beispiel einer nicht weiter optimierbaren Anfrage: Der in die Aggregations-Anfrage eingeschachtelte Filter-Operator kann nicht auf eine höhere Ebene des Anfragebaumes geführt werden. Aufgrund des hohen kalkulatorischen Aufwandes sollten derartige Anfragen vermieden werden.

Theorem 3.5: Anfrageoptimierung: Komposition von Aggregations-Operatoren

Seien Agg_{spec1} und Agg_{spec2} Aggregations-Operatoren, bzw. speziell s- oder t-Normen, und q_i beliebige Sub-Anfragen. Dann kann die folgende Anfrage äquivalent umgeformt werden:

$$Agg_{spec1}(Agg_{spec2}(q_1, ..., q_k), q_{k+1}, ..., q_n) = Agg_{compose(spec1,spec2)}(q_1, ..., q_k, q_{k+1}, ..., q_n)$$

Die Relevanzen des kombinierten Aggregations-Operators berechnen sich dabei analog zu der Anfragestruktur:

$$\mu_{Agg_{compose(X,Y)}}(\langle w_1, \mu_1\rangle, ..., \langle w_k, \mu_k\rangle, \langle w_{k+1}, \mu_{k+1}\rangle, ..., \langle w_n, \mu_n\rangle)$$
$$= \mu_{Agg_X}(\mu_{Agg_Y}(\langle w_1, \mu_1\rangle, ..., \langle w_k, \mu_k\rangle), \langle w_{k+1}, \mu_{k+1}\rangle, ..., \langle w_n, \mu_n\rangle)$$

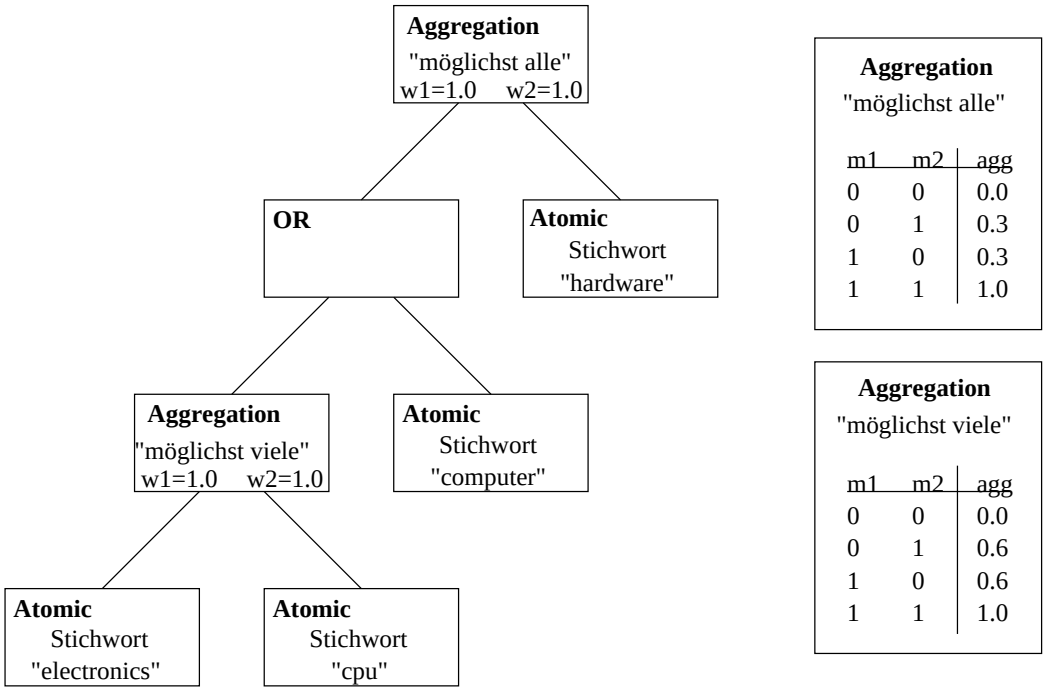


Abbildung 3.13: Beispiel einer nicht optimierten, komplexen Anfrage aus mehreren ineinander verschachtelten Aggregations-Operatoren. Der kalkulatorische Aufwand für die Ausführung einer solchen Anfrage ist ohne Optimierung sehr hoch.

Durch rekursive Anwendung der Gleichung können Aggregations-Operatoren in beliebig vielen Ebenen des Anfragebaums zusammengefasst werden, sofern sie nicht durch eine Ebene mit einem Filter-Operator getrennt sind.

Der Äquivalenz-Beweis für diese Optimierung ist trivial und ergibt sich direkt aus der Analogie der beiden obigen Gleichungen. Anzumerken bleibt allerdings, dass der kombinierte Aggregations-Operator nicht zwangsläufig die Eigenschaften der enthaltenen Operatoren übernimmt. So kann z.B. sogar die wichtige Eigenschaft der Kommutativität verloren gehen. Dies spiegelt die Tatsache wieder, dass die Stichworte der Anfrage in Abbildung 3.13 nicht äquivalent zueinander sind, und dies bei einer äquivalenten Umformung auch bei der Anfrage in Abbildung 3.15 nicht der Fall sein darf. Eine Anfrage, die durch Komposition optimiert wurde, darf also keinen weiteren Optimierungen unterzogen werden, die diese Eigenschaften voraussetzen. Die Komposition muss daher am Ende aller Optimierungen stehen.

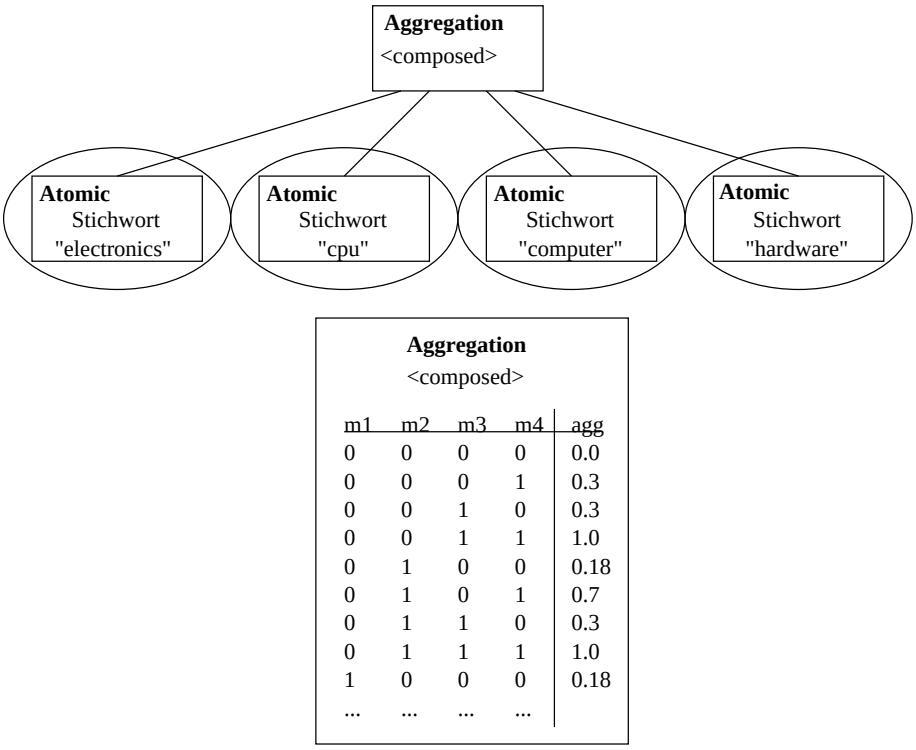
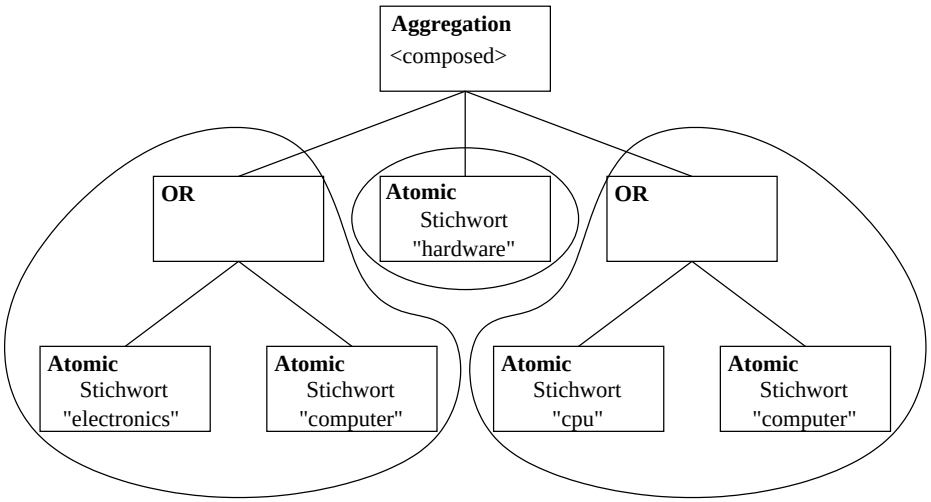


Abbildung 3.14: Erste Form der Optimierung der Anfrage aus Abbildung 3.13. Die ineinander verschachtelten Aggregations-Operatoren sind miteinander kombiniert. Die vier auszuführenden Datenbankanfragen sind durch Ellipsen markiert.



Aggregation
<composed>

m1	m2	m3	agg
0	0	0	0.0
0	0	1	0.18
0	1	0	0.3
0	1	1	0.7
1	0	0	0.18
1	0	1	0.3
1	1	0	0.7
1	1	1	1.0

Abbildung 3.15: Zweite Form der Optimierung der Anfrage aus Abb. 3.13. Im Gegensatz zu der optimierten Anfrage aus Abb. 3.14 sind hier nur noch drei - wieder markierte - Teilanfragen auszuführen, wodurch die gesamte Anfrage trotz höherer Komplexität der Teilanfragen effizienter zu bearbeiten ist.

3.4.5 Optimierungsstrategie

Neben den in den vorherigen Kapiteln 3.4.2-3.4.4 genannten Optimierungen gilt es, durch boolesche Äquivalenz-Umformungen die Voraussetzungen für diese Optimierungen zu schaffen. Diese Standardumformungen, die auf Idempotenz, Distributivität, De Morgan'scher Formel, etc. beruhen, brauchen an dieser Stelle nicht weiter diskutiert werden.

Die Prioritäten der Optimierungen wurden bereits in den jeweiligen Kapiteln angesprochen, so dass sich die folgende komplette Optimierungsstrategie ergibt (funktionale Notation):

Atomare Anfragen werden nicht weiter optimiert:

$$optimize(Atomic_{spec}) \quad = \quad Atomic_{spec}$$

Bei Negationen wird das Negat rekursiv optimiert:

$$optimize(NOT(q)) \quad = \quad NOT(optimize(q))$$

Filter-Anfragen werden nach 3.3 optimiert:

$$optimize(Filter_{spec}(q)) \quad = \quad Filter_{spec}(optimize(AND([q, envelope(spec)])))$$

Aggregations-Anfragen werden nach 3.5 optimiert:

$$optimize(Agg_{spec}([q_1, ..., q_n])) \quad = \quad Agg_{compose(spec,spec_1,...,spec_m)}([b'_1, ..., b'_n, \; a'_{1_1}, ..., a'_{1_{n_1}}, \; ..., \; a'_{m_1}, ..., a'_{m_{n_m}}])$$

mit b'_i boolesch ,
$a'_i = Agg_{spec_i}(a'_{i_1}, ..., a'_{i_{n_i}})$,
$\{q'_i\} = \{b'_i\} \cup \{a'_i\}$ und
$q'_i = optimize(q_i)$

Konjunktive Anfragen werden nach 3.3, 3.4 und 3.5 optimiert:(*)

$$optimize(AND([q_1, ..., q_n])) \quad = \quad \begin{cases} AND([q'_1, ..., q'_n]) & \text{, wenn } q'_i \text{ boolesch } \forall \text{ i} \\ & \text{mit } q'_i = optimize(q_i) \\ Agg_{spec}([AND([a'_1, q'_1, ..., q'_{j-1}, q'_{j+1}, ..., q'_n]), & \text{, wenn } \exists! j \; : \; a' := q'_j = Agg(a_1, ..., a_m) \\ \quad ... & \text{mit } \forall i \neq j \; : \; q'_i \text{ boolesch und} \\ \quad AND([a'_m, q'_1, ..., q'_{j-1}, q'_{j+1}, ..., q'_n])]) & \quad q'_i = optimize(q_i) \\ Agg_{compose(spec_1,...,spec_m)}([AND([a'_{1_1}, b'_1, ..., b'_n]), & \text{, sonst} \\ \quad ... & \text{mit } b'_i \text{ boolesch ,} \\ \quad AND([a'_{1_{n_1}}, b'_1, ..., b'_n]), & \quad a'_i = Agg_{spec_i}(a'_{i_1}, ..., a'_{i_{n_i}}) , \\ \quad ... & \quad \{q'_i\} = \{b'_i\} \cup \{a'_i\} \text{ und} \\ \quad AND([a'_{m_1}, b'_1, ..., b'_n]), & \quad q'_i = optimize(q_i) \\ \quad ... & \\ \quad AND([a'_{m_{n_m}}, b'_1, ..., b'_n])]) & \end{cases}$$

Disjunktive Anfragen werden nach 3.4 und 3.5 optimiert:

$$optimize(OR([q_1, ..., q_n])) \quad = \quad \begin{cases} OR([q'_1, ..., q'_n]) & \text{, wenn } q'_i \text{ boolesch } \forall \text{ i} \\ & \text{mit } q'_i = optimize(q_i) \\ Agg_{spec}([OR([a'_1, q'_1, ..., q'_{j-1}, q'_{j+1}, ..., q'_n]), & \text{, wenn } \exists! j \ : \ a' := q'_j = Agg(a_1, ..., a_m) \\ \quad ... & \text{mit } \forall i \neq j \ : \ q'_i \text{ boolesch und} \\ \quad OR([a'_m, q'_1, ..., q'_{j-1}, q'_{j+1}, ..., q'_n])]) & \quad q'_i = optimize(q_i) \\ Agg_{compose(spec_1,...,spec_m)}([OR([a'_{1_1}, b'_1, ..., b'_n]), & \text{, sonst} \\ \quad ... & \text{mit } b'_i \text{ boolesch ,} \\ \quad OR([a'_{1_{n_1}}, b'_1, ..., b'_n]), & \quad a'_i = Agg_{spec_i}(a'_{i_1}, ..., a'_{i_{n_i}}) \text{ ,} \\ \quad ... & \quad \{q'_i\} = \{b'_i\} \cup \{a'_i\} \text{ und} \\ \quad OR([a'_{m_1}, b'_1, ..., b'_n]), & \quad q'_i = optimize(q_i) \\ \quad ... & \\ \quad OR([a'_{m_{n_m}}, b'_1, ..., b'_n])]) & \end{cases}$$

3.5 Zusammenfassung

In diesem Kapitel wurden zahlreiche Kriterien für Fuzzy-Operatoren, insbesondere für Quantoren aufgestellt und unter besonderer Berücksichtigung der bibliographischen Suche analysiert.

Es wurden mehrere Ansätze von Fuzzy-Quantoren auf diese Kriterien hin untersucht, wobei der WAO_λ-Operator - eine Verallgemeinerung der bekannten p-Norm - als einziger alle geforderten Eigenschaften erfüllt, und darüber hinaus mehrere bekannte Quantoren als Spezialfälle in sich vereint. Durch den Parameter λ kann von einem ODER-Operator, nämlich der bekannten Maximums-Norm über das arithmetische Mittel bis hin zum geometrischen Mittel, einem UND-Operator, gemittelt werden.

Weiterhin wurden Möglichkeiten der Optimierung von Anfragen untersucht, die auf diesen Fuzzy-Operatoren basieren, und es wurde eine Optimierungsstrategie vorgestellt.

Im folgenden Kapitel wird nun untersucht, wie derartig optimierte Fuzzy-Anfragen ausgeführt werden können, d.h. wie große Fuzzy-Mengen effizient über diese Quantoren miteinander verknüpft werden können, und es wird ein Vergleich verschiedener Auswertungsstrategien aufgestellt.

4

Algorithmen zur Aggregation von Fuzzy-Streams

4.1 Motivation und Kapitelübersicht

Nachdem in den vergangenen Kapiteln verschiedene Fuzzy-Operatoren auf ihre Eignung bezüglich des bibliographischen Retrievals hin untersucht wurden, wird in diesem Kapitel der Problemkomplex der effizienten Abarbeitung von Fuzzy-Anfragebäumen, die aus derartigen Operatoren bestehen, behandelt.

Wie bereits einleitend erwähnt, sind die beim gewichteten Retrieval auftretenden Ergebnismengen in der Regel sehr groß und können einen Umfang von mehreren hunderttausend Datensätzen aufweisen (vgl. Kapitel 2.2). Um die Aggregation und das Filtern dieser Mengen mit akzeptablem Antwortzeit-Verhalten durchführen zu können, sind besonders effiziente Auswertungsstrategien nötig, die auf dem beschriebenen Streaming-Prinzip basieren, bei denen bereits nach Einlesen weniger Datensätze Ergebnisse produziert und weitere auf Anfrage hin iterativ berechnet werden können. Dem Benutzer sollen dabei die Ergebnisse mit der höchsten Relevanz zuerst präsentiert werden, es muss eine absteigende Sortierung nach Zugehörigkeitsgrad in Bezug auf die Anfrage sichergestellt werden.

Nach einer genaueren Problembeschreibung und einer Betrachtung der entscheidenden Einflussfaktoren auf das Laufzeitverhalten werden in diesem Kapitel verschiedene Verfahren zur Aggregation und zum Filtern von Fuzzy-Streams untersucht, wobei ihre Voraussetzungen und insbesondere ihr Laufzeitverhalten mit einheitlichen Kostenmaßen verglichen werden. Dabei wird nicht nur speziell das bibliographische Retrieval, sondern generell das gewichtete Retrieval berücksichtigt.

Nach der theoretischen Untersuchung der verschiedenen Algorithmen werden darüber hinaus die Ergebnisse in entsprechenden Praxistests bestätigt.

4.2 Problemstellung und Einflussfaktoren

4.2.1 Problembeschreibung und begleitende Beispiele

Die Berechnung der Relevanzen einer Fuzzy-Menge ergibt sich unmittelbar aus der Definition der Filter- und Aggregations-Operatoren bzw. deren Anwendung auf die einzelnen Elemente der Menge. Um die gewünschten Datenvolumina zu bewältigen, ist es jedoch notwendig, auf das bereits in den vergangenen Kapiteln beschriebene Streaming-Prinzip zurückzugreifen und die Ergebnismenge iterativ zu berechnen, also nach Einlesen möglichst weniger Datensätze bereits Ergebnisse ausgeben zu können. Dabei ist wichtig, dass eine absteigende Sortierung des Streams nach Relevanz gewährleistet ist, da der Benutzer in der Regel nur an wenigen Treffern mit einer hohen Relevanz interessiert ist. Weitere Ergebnisse sollen ggf. auf Anfrage hin berechnet werden.

Die Einbehaltung dieser Sortierung ist der Grund, weswegen die direkte Herangehensweise an die Aggregation von Fuzzy-Mengen nicht geeignet ist: Bei ihr ist ein komplettes Einlesen und Berechnen der Mengen mit einer anschließenden Neusortierung nötig. Nicht nur die Sortierung mit einem Aufwand von $O(N * log(N))$ ist hier ein begrenzender Faktor, insbesondere wird die Anwendung des Streaming-Prinzips durch die Notwendigkeit, die Eingangs-Streams zunächst komplett einzulesen, effektiv verhindert.

Im allgemeinen Fall gilt es also, Verfahren zu finden, die zwei oder mehr nach abfallenden Relevanzen sortierte Eingangs-Streams zu einem ebenfalls nach abfallender Relevanz sortierten Ausgangs-Stream vereinen.

Bei der Untersuchung der Algorithmen werden zwei begleitende Beispiele im Vordergrund stehen:

Als Anwendungsbeispiel für die bibliographische Stichwortsuche dient der Rechercheassistent [57]. In diesem Anwendungs-Szenario ist die gewichtete Fuzzy-Aggregation boolescher bzw. diskret verteilter Attribute (nämlich das Auftreten von Stichworten in verschiedenen Feldern wie „Titel" oder „Abstract") eine übliche Aufgabenstellung. Kombiniert wird dies mit einem „echten", also kontinuierlich verteilten Fuzzy-Attribut „Datum möglichst neu".

Ein Beispiel aus einem anderen Bereich mit echten, kontinuierlich verteilten Fuzzy-Attributen ist die bereits eingangs erwähnte Suche in einer Gebrauchtwagen-Datenbank, in der nach einem möglichst neuen und möglichst günstigen Wagen gesucht wird. Attribute wie die Leistung, der Verbrauch oder der Kilometerstand wären dabei weitere zu berücksichtigende Attribute.

4.2.2 Kostenmaße und Einflussfaktoren

Für die Bewertung der im Folgenden vorgestellten Algorithmen zur Verknüpfung mehrerer Fuzzy-Streams sind die zwei üblichen Kriterien *Speicherverbrauch* und *Laufzeit* als Kostenmaße interessant. Die Laufzeit hängt dabei im Wesentlichen von der Anzahl der ausgeführten Anfragen (in der Regel - bei zwei Attributen und damit zwei Anfragen - einem konstanten Offset) und der Anzahl der I/O-Zugriffe, also üblicherweise der Anzahl der von jedem Eingangs-Stream eingelesenen Objekte ab. Von besonderer, praktischer Bedeutung sind dabei der Speicherverbrauch und die Laufzeit für die ersten n gelieferten Ergebnisse für kleine $n \in [1..50]$ in Abhängigkeit der Größe der Eingangs-Streams N.

Bei den I/O-Zugriffen müssen dabei drei Fälle unterschieden werden: Der sog. „sorted access", welches der sequentielle, geordnete Zugriff über den Stream ist, bildet die Grundlage aller Algorithmen.

Mittels eines „random access" kann über die ID ein Zugriff auf die Relevanz eines Objektes erfolgen, und zwar unabhängig von der durch den Stream festgelegten Reihenfolge. Dieser Zugriff wird nicht von allen Datenbanken unterstützt, kann aber teilweise emuliert werden (vgl. Kapitel 2.3.2.3). Daher verursacht der random access oft höhere Kosten als der sorted access.

Der „direct access" erlaubt ähnlich wie der random access einen direkten Zugriff auf die Relevanz eines Objektes in Bezug auf ein bestimmtes Attribut. Im Gegensatz zum random access kann beim direct access die Relevanz direkt, also ohne Datenbankzugriff sehr schnell *berechnet* werden, ohne dass zusätzliche Kosten anfallen. Dieser Zugriff korrespondiert direkt mit den Filter-Operatoren wie z.B. „Datum möglichst neu", ist aber nicht bei allen Attributen möglich.

Die für *einen* sorted access benötigte Zeit wird als c_s bezeichnet, die für *einen* random access als c_r. Das Verhältnis der Kosten ist $h := \frac{c_r}{c_s}$. Random access und direct access sollen im Folgenden zusammengefasst werden (rechnerisch drückt sich ein direct access durch $c_r = 0$ und damit $h = 0$ aus).

Die im Folgenden verwendeten Kostenmaße sind dementsprechend:

- **sorted accesses** $L_s(N, n)$: Anzahl an sequentiellen Zugriffen auf die zu verknüpfenden Streams
- **random accesses** $L_r(N, n)$: Anzahl an wahlfreien, Index-/ID-basierten Zugriffen auf die Datensätze in der Datenbank
- **Gesamtlaufzeitkosten** $L(N, n)$: Gewichtete Summe von sorted accesses und random accesses: $L(N, n) = L_s(N, n) + h * L_r(N, n)$.
- **Speicherkosten** $M(N, n)$: Anzahl im Speicher zu haltender Datensätze.

Diese Maße hängen jeweils von der Größe der Streams N und der Anzahl an auszugebenden Datensätzen n ab.

Weitere Einflussfaktoren dieser Kostengrößen sind der verwendete Aggregations-Operator und insbesondere die in der Literatur oft vernachlässigte Verteilungsdichte der Relevanzen $g(x, y)$ über die Streams x und y. Insbesondere vier Verteilungen, die im Folgenden näher betrachtet werden, sind hier relevant: die Gleichverteilung, die Normalverteilung, die Exponentialverteilung und die multidimensionale Verteilung.

Für die Berechnung der Kostenmaße werden für die vorgestellten Verfahren die folgenden Integrale und Umkehrfunktionen von $g(x, y)$ benötigt:

$$G_x(a) = \int_0^1 \int_a^1 g(x,y)\, dx\, dy \tag{4.1}$$

$$G_y(a) = \int_0^1 \int_a^1 g(x,y)\, dy\, dx \tag{4.2}$$

$$G(a) = G_x(a) + G_y(a) \tag{4.3}$$

$$A(a) = \int_a^1 \int_{a+1-x}^1 g(x,y)\, dy\, dx \tag{4.4}$$

$$M(a) = \int_a^1 \int_a^1 g(x,y)\, dy\, dx \tag{4.5}$$

Analog wird bei drei zu verknüpfenden Attributen ausgehend von $g(x, y, z)$ benötigt:

$$G_x(a) = \int_0^1 \int_0^1 \int_a^1 g(x,y,z)\, dx\, dy\, dz \tag{4.6}$$

$$G_y(a) = \int_0^1 \int_a^1 \int_0^1 g(x,y,z)\, dx\, dy\, dz \tag{4.7}$$

$$G_z(a) = \int_a^1 \int_0^1 \int_0^1 g(x,y,z)\, dx\, dy\, dz \tag{4.8}$$

$$G(a) = G_x(a) + G_y(a) + G_z(a) \tag{4.9}$$

$$A(a) = \int_a^1 \int_{a+1-z}^1 \int_{a+2-z-y}^1 g(x,y,z)\, dx\, dy\, dz \tag{4.10}$$

$$M(a) = \int_a^1 \int_a^1 \int_a^1 g(x,y,z)\, dx\, dy\, dz \tag{4.11}$$

Zusätzlich werden die Umkehrfunktionen $A^{-1}(A)$ und $M^{-1}(M)$ benötigt. Abbildungen 4.1 und 4.2 visualisieren die Bedeutung dieser Funktionen: $G_x(a)$, $G_y(a)$ und $G_z(a)$ stellen den *Anteil* der von Stream x, y bzw. z eingelesenen Objekte in Abhängigkeit von a, der Relevanz, bis zu der hin die Streams eingelesen wurden, dar (wobei a korrespondierend mit der absteigenden Sortierung gleichzeitig für alle Streams von 1.0 bis 0.0 läuft). $G(a)$ als Summe dieser

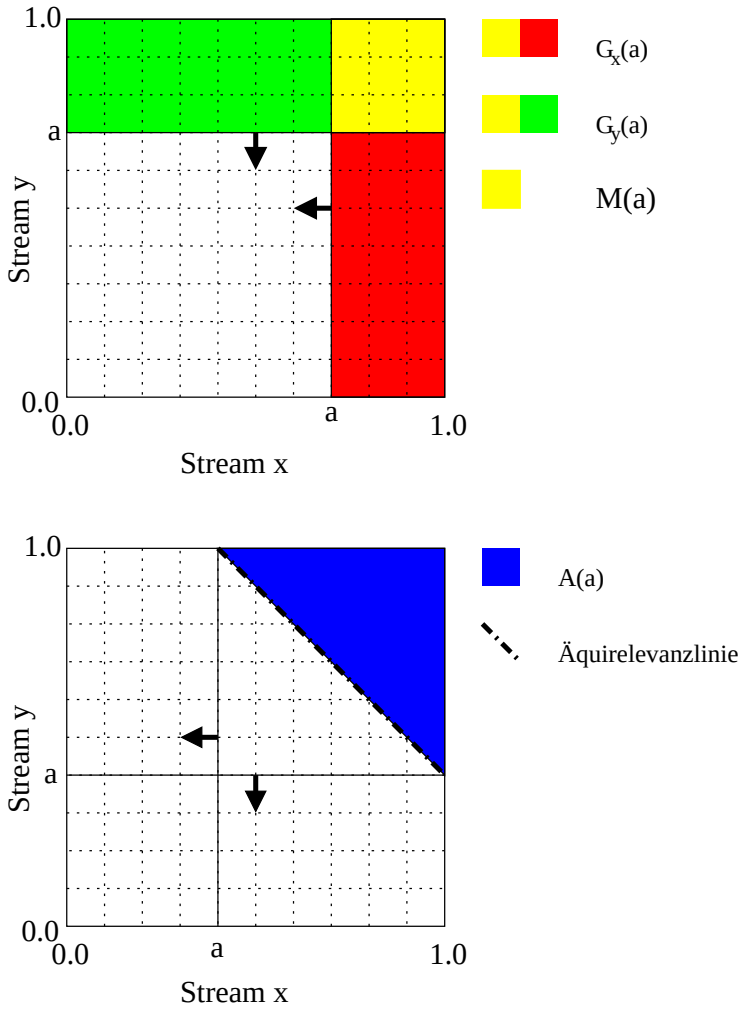


Abbildung 4.1: Die Integrale $G_x(a)$, $G_y(a)$, $M(a)$ und $A(a)$ über die Verteilungsdichte der Streams in Abhängigkeit der Relevanz a. $G_x(a)$ und $G_y(a)$ sind die Anteile der von Stream x bzw. y eingelesenen Datensätze, $M(a)$ der Anteil der matches, also der Datensätze, die bereits von beiden Streams eingelesen wurden. $A(a)$ ist der Anteil der auszugebenden Datensätze und wird durch die Äquirelevanzlinie $\frac{x+y}{2} = \frac{a+1}{2}$ begrenzt.

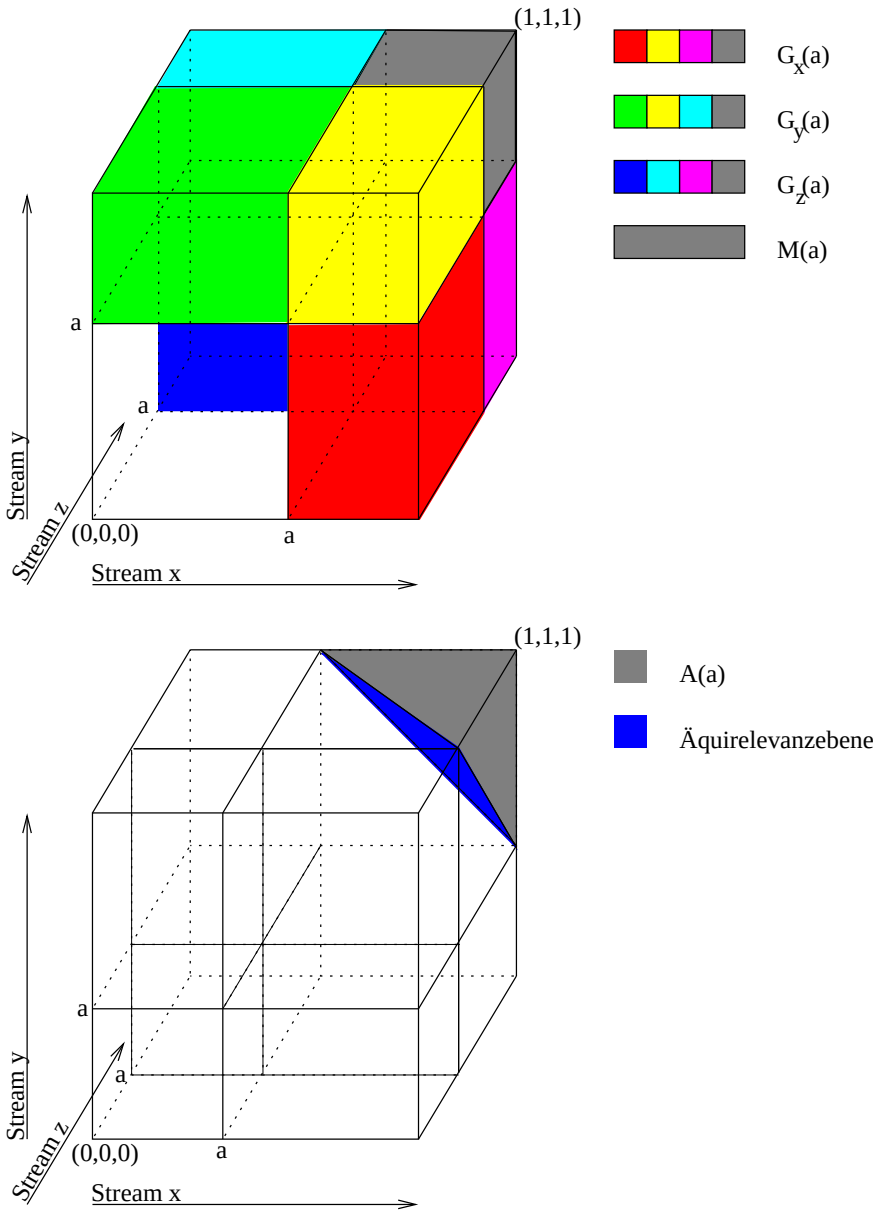


Abbildung 4.2: Die Integrale $G_x(a)$, $G_y(a)$, $G_z(a)$, $M(a)$ und $A(a)$ über die Verteilungsdichte der Streams in Abhängigkeit der Relevanz a. $G_x(a)$, $G_y(a)$ und $G_z(a)$ sind die Anteile der von Stream x, y bzw. z eingelesenen Datensätze, $M(a)$ der Anteil der matches, also der Datensätze, die von allen drei Streams eingelesen wurden. $A(a)$ ist der Anteil der auszugebenden Datensätze und wird durch die Äquirelevanzebene $\frac{x+y+z}{3} = \frac{a+2}{3}$ begrenzt.

Werte läuft daher von 0 bis 2 bzw. 3, da jedes Objekt insgesamt zwei- bzw. dreimal (nämlich einmal pro Stream) eingelesen wird. $M(a)$ ist der Anteil der „matches", also der Objekte, die bereits von allen Streams eingelesen wurden. $A(a)$ ist der Anteil der ausgegebenen Objekte in Abhängigkeit der Eingangs-Relevanz a, wobei als Aggregations-Operator das arithmetische Mittel gewählt wurde. In Abb. 4.1 ist die Äquirelevanz-Linie für die Ausgabe-Relevanz $\mu = \frac{x+y}{2} = \frac{a+1}{2}$ eingezeichnet, die $A(a)$ begrenzt. Analog stellt Abb. 4.2 die sich ergebende Äquirelevanz-Ebene $\mu = \frac{x+y+z}{3} = \frac{a+2}{3}$ dar.

4.2.2.1 Gleichverteilung

Bei der Gleichverteilung ($g(x, y) \equiv 1$) sind beide Attribute unabhängig voneinander und gleichmäßig über das Einheitsintervall verteilt. Abbildung 4.3 zeigt diese Verteilung. Für die Gleichverteilung berechnen sich $G_x(a)$, $G_y(a)$, $G(a)$, $A(a)$ und $M(a)$ sowie deren Inverse trivial:

$$
\begin{aligned}
G_x(a) &= \int_0^1 \int_a^1 1 \; dx \; dy = 1 - a \\
G_y(a) &= \int_0^1 \int_a^1 1 \; dy \; dx = 1 - a \\
G(a) &= G_x(a) + G_y(a) = 2 - 2a \\
A(a) &= \int_a^1 \int_{a+1-x}^1 1 \; dy \; dx = \frac{1}{2} * (1-a)^2 \\
A^{-1}(A) &= 1 - \sqrt{2A} \\
M(a) &= \int_a^1 \int_a^1 1 \; dy \; dx = (1-a)^2 \\
M^{-1}(M) &= 1 - \sqrt{M}
\end{aligned}
$$

4.2.2.2 Negativ korrelierte Normalverteilung

Bei der negativ korrelierten Normalverteilung unterliegen beide Attribute einer Gauß'schen Verteilung über das Einheitsintervall. Sie sind negativ korreliert, d.h. hohe Werte des einen Attributes implizieren niedrige Werte des anderen Attributes und umgekehrt. Als Beispiel dafür dient das bereits genannte Beispiel, bei dem in einer Gebrauchtwagen-Datenbank nach möglichst neuen und möglichst günstigen Autos gesucht werden soll, wobei erfahrungsgemäß nur wenige Fahrzeuge günstig *und* neu, sondern vorwiegend entweder preiswert *oder* neu sind. Eine derartig negativ korrelierte Normalverteilung zweier Attribute ist in Abbildung 4.4 zu sehen.

$$
g(x, y) = F_{norm} * e^{-\frac{1}{2} * (\binom{x}{y} - \vec{\mu})^T K^{-1} (\binom{x}{y} - \vec{\mu})}
$$

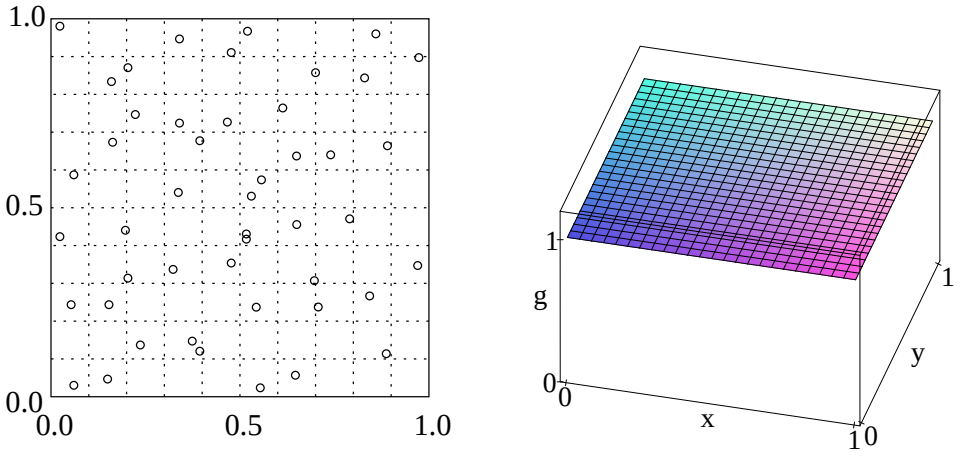


Abbildung 4.3: Gleichverteilung zweier Attribute (links) und Dichtefunktion (rechts)

$$\text{mit } \vec{\mu} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix},\ K = \begin{pmatrix} 1 & k \\ k & 1 \end{pmatrix},\ \text{z.B. } k = -0.7$$

Die gesuchten Integrale und insbesondere ihre Invertierungen sind aufgrund der enthaltenen Exponentialfunktion nicht analytisch lösbar. Daher wurde für die Normalverteilung deren polynomische Approximation

$$g(x,y) \approx F_{norm} * \left(\frac{1}{2} - \frac{1}{2\pi} \left(\left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right)^T K^{-1} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right) \right) + Restglieder \right)$$

mit

$$F_{norm} = \frac{1}{\int_0^1 \int_0^1 \frac{1}{2} - \frac{1}{2\pi} \left(\left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right)^T K^{-1} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right) \right) dx\, dy} = \frac{2(k^2-1)\pi}{\frac{1}{6} - \pi + \pi k^2}$$

verwendet (siehe Anhang B.11). Abbildung 4.4 stellt die Normalverteilung (für den im Folgenden exemplarisch verwendeten Korrelations-Koeffizienten $k = -0.7$) dar. Mit der Approximation ergeben sich für $G_x(a)$, $G_y(a)$, $G(a)$, $A(a)$, $M(a)$ und deren Inverse (vgl. Anhang B.11):

$$\begin{aligned} G_x(a) &\approx 1 - a \\ G_y(a) &\approx 1 - a \\ G(a) &\approx 2 - 2a \\ A(a) &\approx 0.3 * (1-a)^2 \end{aligned}$$

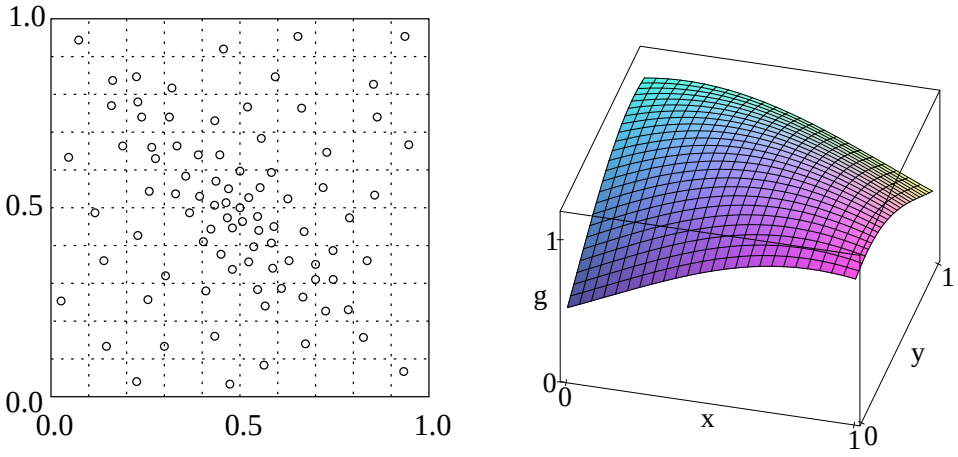


Abbildung 4.4: Normalverteilung negativ korrelierter Attribute (links) und Dichtefunktion (rechts)

$$\begin{aligned} A^{-1}(A) &\approx 1 - \sqrt{\frac{1}{0.3}A} \\ M(a) &= 0.62 * (1-a)^2 \\ M^{-1}(M) &= 1 - \sqrt{\frac{1}{0.62}M} \end{aligned}$$

4.2.2.3 Exponentialverteilung

Eine wie in Abbildung 4.5 zu sehende Verteilung ist bei der Fuzzy-Stichwortsuche häufig anzutreffen. Ein Attribut unterliegt dabei einer Gleichverteilung (z.B. Relevanz eines Dokumentes in Bezug auf den Operator „möglichst neu"), wohingegen das andere Attribut einem exponentiellem Verlauf folgt. Letzterer ist darauf zurückzuführen, dass nur sehr wenige Dokumente alle bzw. fast alle eingegebenen Stichworte enthalten, wohingegen relativ viele Dokumente mindestens eines der eingegebenen Stichworte enthalten. Der Verlauf zwischen diesen beiden Extrema verläuft dabei annähernd exponentiell:

$$\begin{aligned} g(x,y) &= F_{norm} * p^x \\ \text{mit} & \\ F_{norm} &= \frac{ln(p)}{p-1}, \text{ z.B. } p = 0.01 \end{aligned}$$

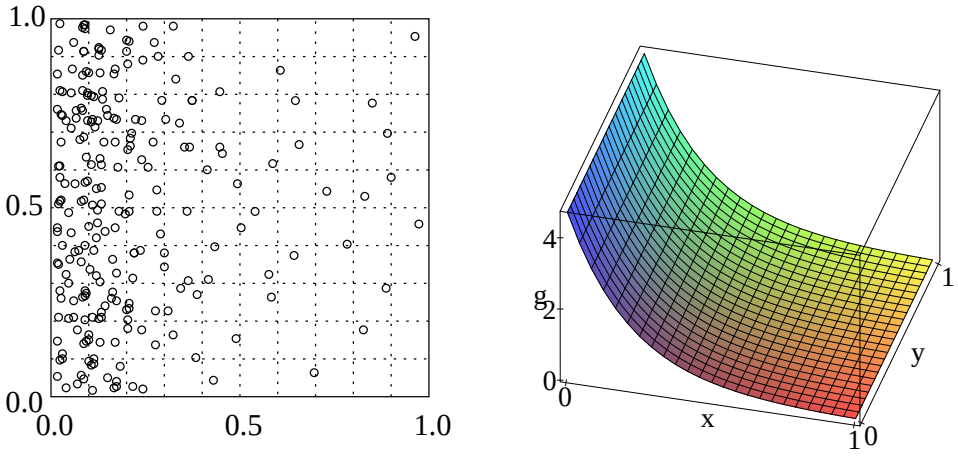


Abbildung 4.5: Exponentialverteilung (links) und Dichtefunktion (rechts)

Analog zur Normalverteilung wird im Weiteren die polynomische Approximation verwendet (s. Anhang B.12):

$$\begin{aligned} g(x,y) &\approx F_{norm} * \left(p + (1-x)^{-2*log_{10}(p)} + Restglieder\right) \\ \text{mit} & \\ F_{norm} &= \frac{1}{\int_0^1 \int_0^1 p + (1-x)^{-2*log_{10}(p)}\, dx\, dy} = \frac{1}{p - \frac{1}{2log_{10}(p)-1}} \end{aligned}$$

Abbildung 4.5 zeigt die Exponentialverteilung für $p = 0.01$. $G(a)$, $A(a)$ und deren Inverse ergeben sich damit zu (vgl. Anhang B.12):

$$\begin{aligned} F_{norm} &= \frac{1}{0.21} \\ G(a) &\approx 1-a \\ A(a) &\approx F_{norm} * p * \frac{1}{2} * (1-a)^2 = \frac{1}{42} * (1-a)^2 \\ A^{-1}(A) &\approx 1 - \sqrt{\frac{2}{F_{norm} * p} * A} = 1 - \sqrt{42A} \\ M(a) &= F_{norm} * p * (1-a)^2 = \frac{1}{21} * (1-a)^2 \\ M^{-1}(M) &= 1 - \sqrt{\frac{M}{F_{norm} * p}} = 1 - \sqrt{21M} \end{aligned}$$

4.2.2.4 Multidimensionale Verteilung

Exemplarisch für mehrdimensionale Verteilungen wird hier eine dreidimensionale Gleichverteilung betrachtet. Die Integrale berechnen sich hier konkret als:

$$
\begin{aligned}
G_x(a) &= \int_0^1 \int_0^1 \int_a^1 1 \; dx \; dy \; dz = 1 - a \\
G_y(a) &= \int_0^1 \int_0^1 \int_a^1 1 \; dy \; dx \; dz = 1 - a \\
G_z(a) &= \int_0^1 \int_0^1 \int_a^1 1 \; dz \; dx \; dy = 1 - a \\
G(a) &= G_x(a) + G_y(a) + G_z(a) = 3 - 3a \\
A(a) &= \int_a^1 \int_{a+1-z}^1 \int_{a+2-z-y}^1 1 \; dx \; dy \; dz = \frac{1}{6} * (1-a)^3 \\
A^{-1}(A) &= 1 - \sqrt[3]{6A} \\
M(a) &= (1-a)^3 \\
M^{-1}(M) &= 1 - \sqrt[3]{M}
\end{aligned}
$$

4.3 Untersuchung verschiedener Algorithmen zur Aggregation

Nachdem in den vergangenen Kapiteln die entscheidenden Kostenmaße definiert und ihre Haupt-Einflussfaktoren festgestellt wurden, können nun die verschiedenen Verfahren zur Aggregation von Fuzzy-Streams, die teilweise aus der Literatur bekannt sind, darauf aufbauend untersucht werden. Neben speziellen Verfahren zur Aggregation boolescher oder diskret verteilter Attribute wird der Schwerpunkt auf den allgemeinen Verfahren zur Aggregation echter Fuzzy-Attribute liegen.

Die Untersuchung wird dabei jeweils aus einer informellen Beschreibung des Algorithmus (meistens unterstützt durch ein veranschaulichendes Beispiel), dem formalen Pseudocode und einer theoretischen Aufwandsabschätzung bestehen.

In einem abschließenden Kapitel werden die Aufwandsabschätzungen durch entsprechende Praxistests bestätigt.

4.3.1 Aggregation boolescher oder diskreter Attribute

Unter der Voraussetzung, dass die zu aggregierenden Attribute boolesch oder zumindest diskret sind, kann eine Anfrage aus einer Aggregation, wie bereits aus der Literatur bekannt, auf sehr effiziente Art und Weise bearbeitet werden.

Diese Bedingung erscheint z.B. bei der Stichwortsuche zunächst selbstverständlich, ist aber nicht notwendigerweise gegeben. Volltextdatenbanken können beispielsweise die Auftrittswahrscheinlichkeit bzw. die relative Häufigkeit eines Stichwortes bewerten und in die Relevanz-Bewertung einbeziehen. Auch Wortstammbildung oder Komposita-Zerlegung können dazu verwendet werden, dass bestimmte Dokumente auf ein Stichwort zu einem stärkeren Grad „matchen" als andere Dokumente.

Für rein boolesche Datenbanken, die lediglich bewerten, *ob* ein Stichwort in einem Dokument auftritt oder nicht, zerfällt die Ergebnismenge einer Aggregations-Anfrage wie z.B. „möglichst_viele(fuzzy,retrieval,bool)" jedoch in einzelne *Klassen* von Ergebnissen, die keine kontinuierliche, sondern eine diskrete Verteilung der Relevanzen aufweisen und daher sehr effizient zu berechnen sind:

Bei s Stichworten gibt es - da es für jedes Stichwort 2 Möglichkeiten (im Dokument enthalten / nicht im Dokument enthalten) gibt - insgesamt $k = 2^s$ Kombinationen des Auftretens von Stichworten in den Dokumenten. Also setzt sich die Ergebnismenge aus $k = 2^s$ Klassen von Dokumenten mit jeweils gleicher Relevanz zusammen (vgl. Abb. 4.6): Dokumente, in denen *alle* Stichworte vorkommen (in diesem Fall alle Dokumente, die die Datenbank bei der [booleschen] Anfrage „fuzzy AND retrieval AND bool" liefert), erhalten eine Relevanz von 1.0 und werden zuerst ausgegeben. Danach folgen die Dokumente, in denen alle bis auf das unwichtigste Stichwort (mit dem niedrigsten, vom Benutzer zugewiesenen Gewicht) vorkommen (in diesem Fall alle Dokumente, die für die boolesche Anfrage „fuzzy AND retrieval AND [NOT bool]" geliefert werden), usw.

Analog zerfällt die Menge aller Dokumente bei s eingegebenen Stichworten in $k = (a + 1)^s$ Klassen, wenn es a verschiedene Attribute (wie z.B. Titel und Volltext) gibt, in denen ein Stichwort vorkommen kann. Dabei wird, um die Komplexität nicht noch stärker wachsen zu lassen, davon ausgegangen, dass die mit einem Stichwort korrespondierenden, einzelnen Klassen nicht unabhängig voneinander sind, sondern dass es eine durch eine Inklusion gegebene, natürliche Ordnung auf den Attributen gibt. Das heißt, dass ein Dokument eine niedrige Relevanz bekommt, wenn ein Stichwort nur im Volltext auftaucht und eine hohe Relevanz, wenn das Stichwort im Titel vorkommt - in diesem Fall allerdings unabhängig davon, ob es zusätzlich auch im Volltext vorkommt (was sehr wahrscheinlich ist) oder nicht.[1] Dieser Fall ist für $s = 2$ Stichworte in Abbildung 4.7 dargestellt. Für mehr Stichworte ist eine vollständige und sinnvolle Darstellung kaum mehr möglich. Im Folgenden wird von booleschen Attributen (also $k = 1$) ausgegangen.

Als formales Effizienzmaß der Anfrage-Bearbeitung ist das bereits in den vergangenen Kapiteln verwendete Maß, nämlich die Anzahl an auszugebenden Dokumenten in Abhängig-

[1] Geht man davon aus, dass die Attribute nicht nach dem Inklusions-Prinzip geordnet sind, sondern unabhängig voneinander, so erhöht sich der Aufwand auf $k = (2^a)^s = 2^{(a*s)}$ Ergebnisklassen.

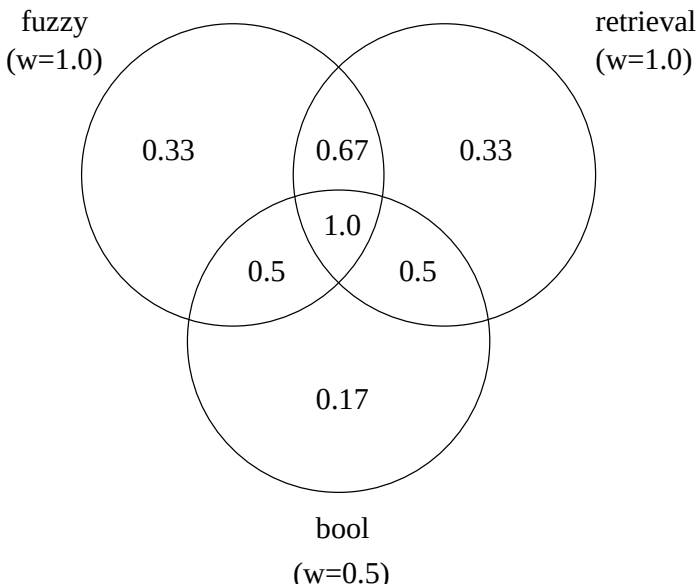


Abbildung 4.6: Die Ergebnisklassen einer Aggregations-Anfrage aus drei booleschen Teilanfragen mit ihren Relevanzen. Zwei mal zwei Klassen haben jeweils die gleiche Relevanz und können daher zu einer Klasse zusammengefasst werden.

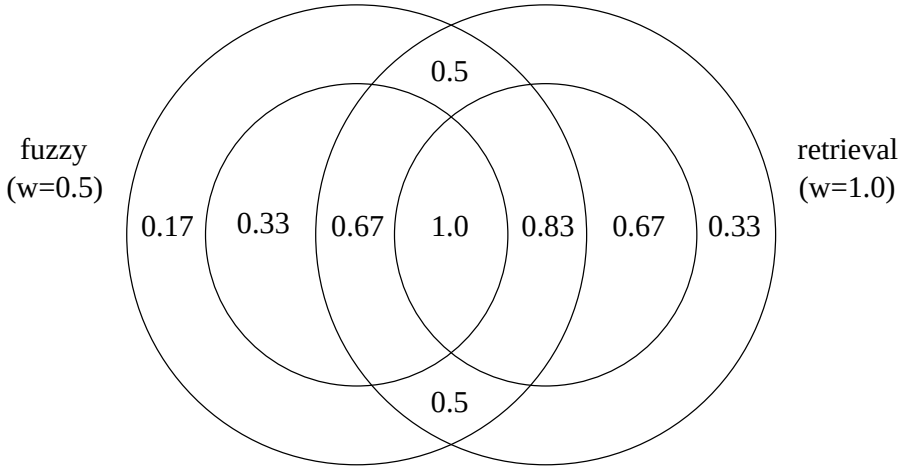


Abbildung 4.7: Die Ergebnisklassen einer Aggregations-Anfrage aus zwei diskreten, 3-wertigen Teilanfragen mit ihren Relevanzen. Zwei mal zwei Klassen haben jeweils die gleiche Relevanz und können daher zu einer Klasse zusammengefasst werden.

keit der Anzahl an eingelesenen Dokumenten nicht geeignet, da jedes eingelesene Dokument umgehend auf den Ausgabe-Stream geleitet werden kann und an dieser Stelle kein Overhead entsteht.

Der die Performance limitierende Faktor ist hier die *Anzahl (und Komplexität) der zu formulierenden booleschen Teilanfragen*, die die Datenbank ausführen muss, also die Anzahl der Ergebnisklassen. Sie hängt im Wesentlichen von der Anzahl der eingegebenen Stichworte, aber auch von der Größe der Datenbanken und der Stichwort-Auftrittswahrscheinlichkeit ab. Dies wird im Folgenden formal untersucht:

Sei N die Größe der Datenbank, gemessen in Anzahl an Datensätzen, s die Anzahl der gesuchten Stichworte und p die Wahrscheinlichkeit, dass ein Stichwort in einem Dokument vorkommt (vereinfachend sei angenommen, die Stichworte seien statistisch unabhängig und die Auftrittswahrscheinlichkeit p sei für alle eingegebenen Stichworte gleich). Dann berechnet sich $d_{N,s,p}(i) \quad (i = 1..(2^s - 1))$, der Erwartungswert für die Anzahl der Dokumente, die bei der i-ten Teilanfrage geliefert werden, als:

$$d_{N,s,p}(i) = N * p^{s-n_{i,s}} * (1-p)^{n_{i,s}}$$

Dabei ist $n_{i,s}$ die Anzahl der in der i-ten Teilanfrage vorkommenden, negierten Stichworte, und berechnet sich wie folgt (ein Beispiel dazu zeigt Tabelle 4.1):

$$n_{i,s} = argmin_k \left[(\sum_{j=0}^{k} \binom{s}{k}) \geq i \right]$$

Für die Performance-Abschätzung interessant ist nun die Summe $D(q)$ der $d(i)$, also die Anzahl der insgesamt gelieferten Dokumente, nachdem q Anfragen ausgeführt wurden (vgl. ebenfalls Tabelle 4.1):

$$\begin{aligned} D_{N,s,p}(q) \quad &:= \sum_{i=1}^{q} d_{N,s,p}(i) && q = 1..(2^s - 1) \\ &= \sum_{i=1}^{q} N * p^{s-n_{i,s}} * (1-p)^{n_{i,s}} \\ &= N * \sum_{i=1}^{q} p^{s-n_{i,s}} * (1-p)^{n_{i,s}} \end{aligned}$$

Diese Funktion (genauer: ihre Umkehrfunktion $D^{-1}_{N,s,p}$, die die Anzahl an auszuführenden Teilanfragen in Abhängigkeit der Anzahl an angeforderten Dokumenten angibt) drückt nun das gesuchte Aufwandsmaß aus, welches noch von den Parametern N, s und p abhängig ist. Da die Umkehrfunktion nur aufwändig auf numerische Weise ermittelt werden kann, wird im Folgenden die Originalfunktion $D_{N,s,p}(q)$ untersucht:

Zunächst fällt auf, dass die Anzahl an insgesamt gelieferten Dokumenten nach Ausführung der q-ten Anfrage $D_{N,s,p}(q)$ direkt proportional zu N, der Größe der Datenbank ist. Das hat zur Folge, dass der Algorithmus bei größeren Datenbanken früher terminiert: Obwohl bei einem größeren zu durchsuchenden Datenvolumen der Aufwand pro Teilanfrage datenbankseitig höher ist, wird dies durch eine geringere Anzahl an Teilanfragen kompensiert, so dass der Gesamtaufwand bei steigender Datenbankgröße gleich bleibt bzw. sogar sinken kann.

Abbildung 4.8 zeigt die Anzahl an gelieferten Ergebnissen $D_{N,s,p}(q)$ in Abhängigkeit der Stichwort-Auftrittswahrscheinlichkeit p und der Anzahl an ausgeführten Teilanfragen q für

i	Teilanfrage	$n_{i,4}$	$d_{1000000,4,0.01}(i)$	$D_{1.000.000,4,0.01}(i)$
1	$A \wedge B \wedge C \wedge D$	0	0.01	0.01
2	$A \wedge B \wedge C \wedge \neg D$	1	0.99	1.00
3	$A \wedge B \wedge \neg C \wedge D$	1	0.99	1.99
4	$A \wedge \neg B \wedge C \wedge D$	1	0.99	2.98
5	$\neg A \wedge B \wedge C \wedge D$	1	0.99	3.97
6	$A \wedge B \wedge \neg C \wedge \neg D$	2	98.01	101.98
7	$A \wedge \neg B \wedge C \wedge \neg D$	2	98.01	199.99
...	...	...	...	...
15	$\neg A \wedge \neg B \wedge \neg C \wedge D$	3	9702.99	39404.00

Tabelle 4.1: Teilanfragen und gelieferte Dokumente bei einer Anfrage „möglichst viele A, B, C, D" (s=4) bei einer Datenbank mit $N = 1.000.000$ Dokumenten und einer Auftrittswahrscheinlichkeit von $p = 0.01$.

eine mittelgroße Datenbank mit $N = 1.000.000$ Dokumenten jeweils für Anfragen mit $s = 3$, 4, 5 und 6 Stichworten. Neben dem polynomischen Zusammenhang zwischen p und D (je allgemeiner die eingegebenen Stichworte sind, desto mehr relevante Dokumente werden bereits frühzeitig geliefert und desto weniger Teilanfragen müssen ausgeführt werden) ist hier der entscheidende Einfluss 2^s von der Anzahl der eingegebenen Stichworte s zu erkennen.

Eine Beschränkung dieses exponentiellen Zusammenhangs mit s ergibt sich aus der Tatsache, dass einige Ergebnisklassen aufgrund gleicher Relevanzen zusammenfallen. So können in dem eingangs genannten Beispiel (Abb. 4.6) die Teilanfragen

```
fuzzy AND (NOT retrieval) AND bool
```

und

```
(NOT fuzzy) AND retrieval AND bool
```

zu *einer* Teilanfrage

```
   (fuzzy AND (NOT retrieval) AND bool)
OR ((NOT fuzzy) AND retrieval AND bool)
```

verbunden werden, da beide Anfragen Dokumente mit einer Relevanz von 0.5 liefern. Diese kann vereinfacht werden zu

```
(fuzzy XOR retrieval) AND bool
```

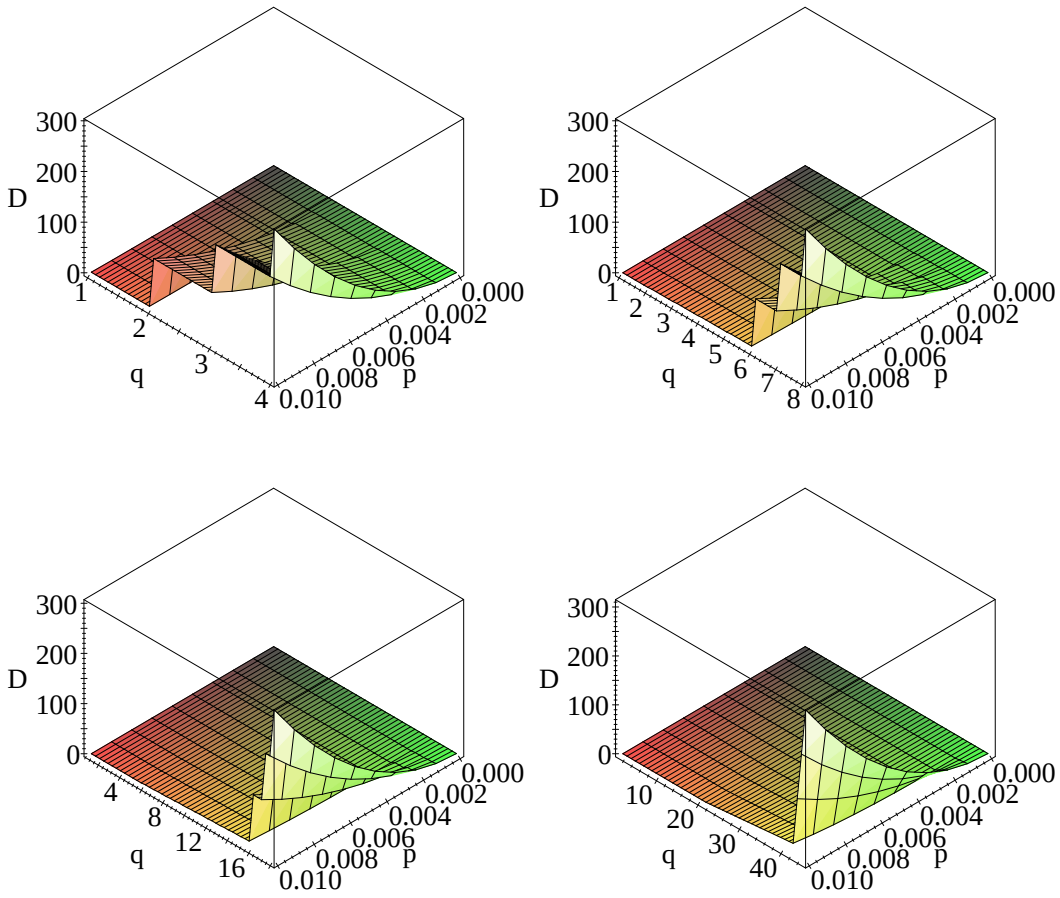


Abbildung 4.8: Anzahl der gelieferten Dokumente für eine Datenbank der Größe $N = 1.000.000$ und (von oben links nach unten rechts) $s = 3, 4, 5$ und 6 Stichworte in Abhängigkeit der Stichwort-Auftrittswahrscheinlichkeit p und der Anzahl an ausgeführten Teilanfragen q.

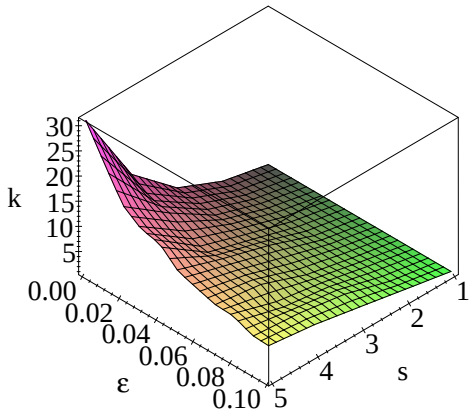


Abbildung 4.9: Anzahl der verschiedenen Dokumentklassen k in Abhängigkeit der Anzahl der eingegebenen Stichworte s und der Granularität der Relevanz ϵ. (Heuristisch ermittelter Graph.)

Im Extremfall, wenn alle Stichworte dasselbe Gewicht haben, hängt das Gesamt-Ranking nur noch von der *Anzahl* der enthaltenen Stichworte ab, so dass die eigentlich $k = 2^s$ Klassen in $k = s$ Klassen zusammenfallen. Im Folgenden wird von einer Quantisierung mit der Granularität ϵ ausgegangen, d.h. jede Klasse enthält Datensätze, die in ihrer Relevanz maximal ϵ auseinanderliegen. Geht man von gleichverteilten Gewichten aus, so spielt der Effekt des Zusammenfallens von Klassen kaum eine Rolle, sofern die Anzahl der Stichworte relativ klein bleibt ($s \leq 4 \; bis \; 5$) und ϵ ebenfalls klein ist. Abbildung 4.9 zeigt die Anzahl k der Dokumentklassen in Abhängigkeit der beiden Faktoren Stichwortanzahl s und der Relevanz-Granularität ϵ. Tatsächlich wird der exponentielle Zusammenhang zwischen s und k für $s \leq 4 \; bis \; 5$ durch $\epsilon = 0.01$ kaum gestört. Die maximale Obergrenze der Dokumentklassen liegt bei $1.0/\epsilon$.

Unter Berücksichtigung der beiden Faktoren Stichwortanzahl s und Zusammenfallen von Ergebnisklassen durch ϵ ergibt sich - gemessen in Anzahl an maximal auszuführenden Teilanfragen k - ein gesamtes „worst case"-Laufzeitverhalten von $O(min(2^s, 1/\epsilon))$. In der Praxis ist k jedoch deutlich niedriger und nähert sich nur für große s der oberen Schranke an, vgl. Abb. 4.10. Außerdem kann durch eine iterative Recherchestrategie, in der der Benutzer nach und nach Stichworte in seine Anfrage integriert, ein akzeptables Antwortzeit-Verhalten sichergestellt werden.

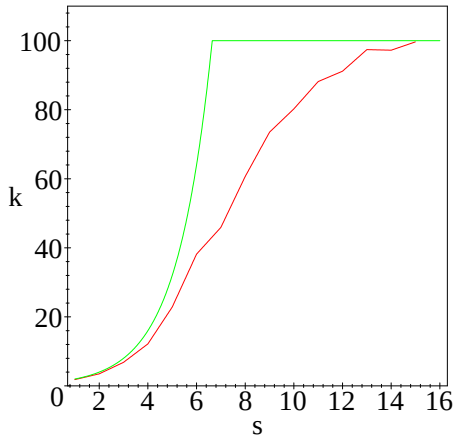


Abbildung 4.10: Anzahl der verschiedenen Dokumentklassen k insgesamt in Abhängigkeit der Anzahl der eingegebenen Stichworte s bei einer Granularität der Relevanz von $\epsilon = 0.01$ (rot: heuristisch ermittelter Graph, grün: obere Schranke $min(2^s, 1/\epsilon)$)

4.3.2 Range-Queries-Algorithmus (RQA)

Glagowski et al schlagen in [24] vor, eine Anfrage, für die mehrere Fuzzy-Attribute verknüpft werden sollen, in Anfrageklassen, die bestimmte Bereiche der Ergebnismenge abdecken, sog. Range-Queries, zu zerteilen.

Jedes Attribut wird dabei in Intervalle zerlegt, so dass deren Schnitt jeweils einen quadratischen, bzw. in höheren Dimensionen kubischen/hyperkubischen Ausschnitt des Ergebnisraumes abdeckt, für den die Relevanzen der Objekte berechnet werden können. Ergebnisse, die über einer entsprechenden Grenz-Relevanz liegen, können dann ausgegeben werden. Dies wird anhand des Beispiels einer Gebrauchtwagen-Datenbank verdeutlicht.

Um eine nach Relevanz absteigend sortierte Ergebnismenge zu erhalten, wird zunächst der Bereich $[0.9..1.0]$ für die Gesamt-Relevanz betrachtet und für diesen der Bereich berechnet, in dem die Zugehörigkeitsgrade der beiden einzelnen Fuzzy-Mengen („möglichst günstig" und „möglichst neu") liegen müssen. Verwendet man als Aggregations-Operator das arithmetische Mittel ($\frac{\mu_A+\mu_B}{2}$), so ergibt sich für μ_A und μ_B das Intervall $[0.8..0.1]$.

Daraus lassen sich - durch die Umkehrfunktionen der beiden Filter-Operatoren - die oberen Schranken für die Attribute „Preis" und „Baujahr" ermitteln. Je nach Implementation der Filter-Operatoren ergeben sich z.B. die oberen Grenze $1000\$$ und Baujahr 1995.

Die an die Datenbank gestellte (boolesche) Anfrage nach allen Fahrzeugen mit einem Baujahr nach 1995 und einem Preis von weniger als $1000\$$ liefert z.B. die folgende Ergebnismen-

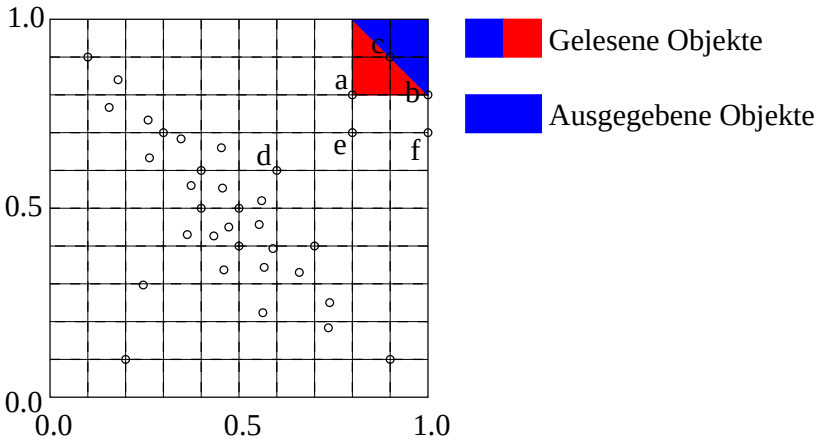


Abbildung 4.11: Zustandsaufnahme nach Ausführung der ersten Range-Query

ge (wobei hier bereits der Preis und das Baujahr in die entsprechenden Zugehörigkeitsgrade μ_A bzw. μ_b umgerechnet ist):

$$\{a(\mu_A = 0.8, \mu_B = 0.8), b(\mu_A = 1.0, \mu_B = 0.8), c(\mu_A = 0.9, \mu_B = 0.9)\}$$

Für diese Ergebnisse werden nun die Gesamt-Relevanzen der Aggregation berechnet:

$$\{a(\mu_{agg} = 0.8), b(\mu_{agg} = 0.9), c(\mu_{agg} = 0.9)\}$$

Von diesen 3 Ergebnissen liegen jedoch nur 2 tatsächlich im anfangs festgelegten Intervall und können somit an den Benutzer ausgegeben werden. Abbildung 4.11 visualisiert diesen aktuellen Zustand.

Die Berechnung der weiteren Treffer erfolgt durch einen neuen Iterations-Durchgang für das Intervall $[0.8..0.9]$ für die Gesamt-Relevanz. Für die einzelnen Zugehörigkeitsgrade ergibt sich dabei ein Intervall von $[0.6..1.0]$. Die Ergebnismenge für die entsprechende boolesche Anfrage wäre z.B.:

$$\begin{aligned}\{&a(\mu_A = 0.8, \mu_B = 0.8), b(\mu_A = 1.0, \mu_B = 0.8), c(\mu_A = 0.9, \mu_B = 0.9),\\ &d(\mu_A = 0.6, \mu_B = 0.6), e(\mu_A = 0.8, \mu_B = 0.7), f(\mu_A = 1.0, \mu_B = 0.7)\}\end{aligned}$$

Dabei sind die Datensätze a, b und c zum zweiten Mal von der Datenbank angefordert worden, was jedoch nicht vermeidbar ist, da z.B. Datensatz a, der zunächst nicht ausgege-

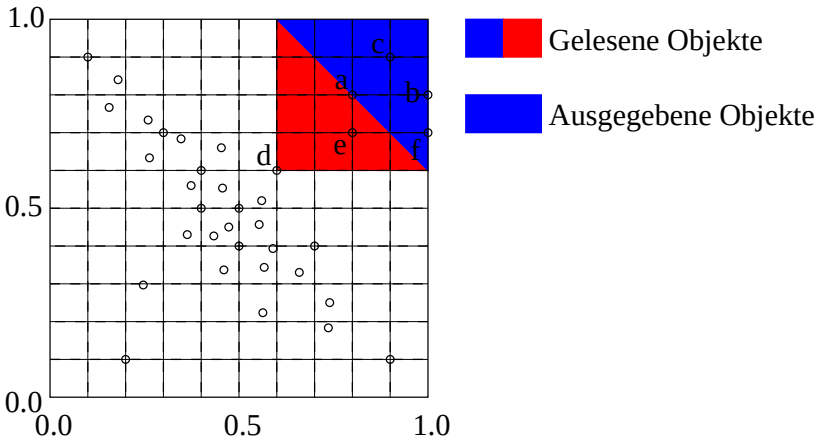


Abbildung 4.12: Zustandsaufnahme nach Ausführung der zweiten Range-Query

ben werden durfte, in diesem Iterations-Schritt nun doch ausgegeben wird. Die Gesamt-Relevanzen der Aggregation berechnen sich wie folgt (wobei hier die bereits ausgegebenen Datensätze b und c nicht weiter berücksichtigt werden):

$$\{a(\mu_{agg} = 0.8), d(\mu_{agg} = 0.6), e(\mu_{agg} = 0.75), f(\mu_{agg} = 0.85)\}$$

Von diesen Datensätzen liegen nur a und f im festgelegten Intervall $[0.8..0.9]$ und können an den Benutzer ausgegeben werden. Jetzt ist auch deutlich, warum Datensatz a nicht bereits im vorigen Iterations-Schritt ausgegeben werden konnte und in diesem Schritt erneut angefordert werden musste, da der Datensatz f eine höhere Gesamt-Relevanz aufweist. Dieser Zustand wird in Abbildung 4.12 gezeigt.

Die Iteration über die Intervalle wird weiter fortgesetzt, bis die gewünschte Anzahl an Datensätzen geliefert worden ist.

Die Nachteile dieses Verfahrens wurden teilweise bereits im angegebenen Beispiel deutlich:

1. Die Datenbank muss die Möglichkeit der intervallbasierten Anfragen unterstützen. Es genügt nicht, wenn sie Ergebnisse nach den einzelnen Relevanzen sortiert ausgeben kann. Dabei muss sie die Relevanzen beider Attribute bei der Ausgabe mitliefern (direct access), damit die Gesamt-Relevanz berechnet werden kann. Andernfalls sind zusätzliche random accesses notwendig.
2. Obwohl die Bereiche für die Gesamt-Relevanzen der Aggregation selbstverständlich disjunkt festgelegt werden, überschneiden sich von Iteration zu Iteration die Intervalle für die Zugehörigkeitsgrade der einzelnen Attribute erheblich, wodurch es sich nicht

umgehen lässt, dass (gerade bei mehreren Iterations-Durchgängen) viele Datensätze mehrfach angefordert werden: Insbesondere liegt die Obergrenze für den einzelnen Zugehörigkeitsgrad fast immer bei 1.0. Dies kommt einer Neuberechnung des gesamten, bereits berechneten Ergebnisses in jedem Iterations-Schritt gleich.

3. Werden keine Datensätze mit hoher Relevanz gefunden, oder werden viele Datensätze vom Benutzer angefordert, so läuft die Iteration soweit, dass das Intervall für die Gesamt-Relevanz den Wert 0.5 beinhaltet, und das Verfahren bricht (bei Verwendung des arithmetischen Mittels als Aggregationsoperator) vollständig zusammen: Als Untergrenze für die Zugehörigkeitsgrade ergibt sich 0.0, als Obergrenze 1.0, also das komplette Einheitsintervall. Es werden dann *alle* Datensätze der Datenbank angefordert, was dem „brute force"-Verfahren gleichkommt. Bei Verwendung des max-Operators stellt sich dieses Problem grundsätzlich bereits zu Beginn.

4. Die Zerlegung des Einheitsintervalls in Teilintervalle für die Gesamt-Relevanz ist nicht zufriedenstellend gelöst. Hier können sehr unterschiedliche Granulierungen - je nach Größe der Datenbank, Art der ausgewählten Aggregations- und Filter-Operatoren und der Beschaffenheit der Datenbasis - sinnvoll sein.

Diese Nachteile sind vor allem darauf zurückzuführen, dass das Verfahren nicht auf Ergebnis-*Streams*, sondern auf ganzen Ergebnis-*Mengen* arbeitet. Auf eine detailliertere Performance-Analyse wird daher an dieser Stelle verzichtet, auch weil die im Folgenden vorgestellten Verfahren - insbesondere der Discrete-Streams-Algorithmus aus Kapitel 4.3.8 - diese Probleme durch eine echte Streaming-Herangehensweise umgehen und somit ein deutlich besseres Skalierungsverhalten aufweisen.

4.3.3 Pfeifer-Fuhr-Algorithmus (PFA)

Pfeifer und Fuhr entwickeln in [56] eine Strategie zur Verknüpfung von zwei oder mehr Fuzzy-Streams, und verdeutlichen diese am Beispiel der einleitend vorgestellten Anfrage an eine Gebrauchtwagen-Datenbank mit zwei Attributen.

Bei dem beschriebenen Verfahren werden von beiden jeweils absteigend nach Relevanz sortierten Streams parallel Datensätze eingelesen, bis bei beiden Streams eine *„Grenzwert-Relevanz"* von a erreicht wird. Sobald ein Datensatz in beiden Streams aufgetaucht ist, kann seine Gesamt-Relevanz berechnet werden.

Gleichzeitig verringert sich mit fortschreitendem Einlesen der Streams auch die *maximale Relevanz*, welche ein noch nicht von beiden Streams eingelesener Datensatz haben kann: Ein Datensatz, welcher z.B. von dem ersten Stream bereits eingelesen wurde (und dort maximal eine Relevanz von 1.0 haben kann), kann auf dem zweiten Stream, auf dem er noch nicht gelesen wurde, maximal eine Relevanz von a haben, so dass als Gesamt-Relevanz (bei Verwendung des arithmetischen Mittels) maximal $(1.0 + a)/2$ möglich ist (vgl. Grenzbedingung Def. 3.15.8).

Demnach kann zu einem Zeitpunkt, zu dem beide Streams bis zu einer Relevanz von a eingelesen sind, jeder Datensatz, der eine Gesamt-Relevanz von mindestens $(1.0 + a)/2$ hat, auf den Ausgabe-Stream geleitet werden. Alle anderen Datensätze müssen in einem Puffer verbleiben. In Pseudocode formuliert, lautet der Algorithmus wie folgt:

```
INPUT: Stream s1, s2;
OUTPUT: Stream out;

List s1Buffer, s2Buffer;
SortedSet outBuffer; // sortiert nach Relevanz
Dataset d;

WHILE NOT out.closed() DO

  // naechsten Datensatz lesen und Relevanz fuer matches berechnen:
  IF s1.currentRelevance>s2.currentRelevance THEN
    d:=s1.read();
    IF s2Buffer.contains(d) THEN
      d.relevance=agg(d.relevance,s2Buffer.get(d).relevance);
      s2Buffer.remove(d);
      outBuffer.add(d);
    ELSE
      s1Buffer.add(d);
    END IF
  ELSE
    d:=s2.read();
    IF s1Buffer.contains(d) THEN
      d.relevance=agg(d.relevance,s1Buffer.get(d).relevance);
      s1Buffer.remove(d);
      outBuffer.add(d);
    ELSE
      s2Buffer.add(d);
    END IF
  END IF

  // Datensaetze mit einer Relevance groesser als
  // "threshold-relevance" auf Outputstream schreiben:
  FOR EACH d IN outBuffer WITH d.relevance>=
      agg(max(s1.currentRelevance,s2.currentRelevance),1) DO
    out.write(d);
    outBuffer.remove(d);
  END FOR

END WHILE
```

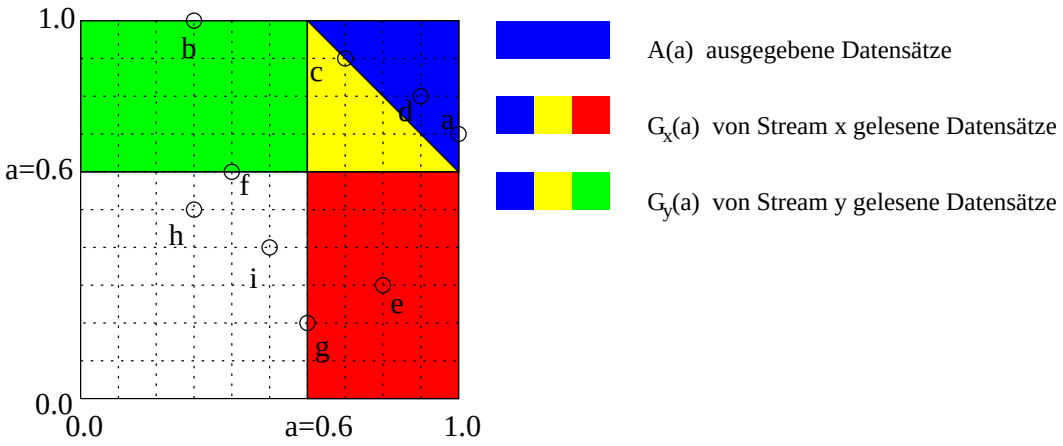


Abbildung 4.13: Verteilung von Datensätzen über zwei Fuzzy-Streams. Zustandsaufnahme für Grenz-Relevanz $a = 0.6$ mit Äquirelevanz-Linie für $\mu = 0.8$.

Zu der Ausführung ein Beispiel: Abbildung 4.13 zeigt eine Verteilung von Datensätzen über zwei Fuzzy-Streams. Insgesamt haben die Datensätze hier eine im Mittel leicht überdurchschnittliche Relevanz - trotzdem, so wird sich zeigen, muss von den Streams relativ weit im Voraus eingelesen werden, bis überhaupt Datensätze auf den Ausgabe-Stream geleitet werden können:

Stream 1: ID(Relevanz)

a(1.0)	d(0.9)	e(0.8)	c(0.7)	g(0.6)	i(0.5)	f(0.4)	...

Stream 2: ID(Relevanz)

b(1.0)	c(0.9)	d(0.8)	a(0.7)	f(0.6)	h(0.5)	i(0.4)	...

In den ersten beiden Schritten, in denen die jeweils ersten beiden Datensätze von den Streams eingelesen werden, kann noch kein Ergebnis ausgegeben werden, weil es noch zu keinen matches, also Überschneidungen gekommen ist und daher noch keine Gesamt-Relevanz berechnet werden kann.

Im dritten Schritt wird von Stream 1 Datensatz e und von Stream 2 Datensatz d gelesen, der auch bereits in Stream 1 auftauchte. Für diesen kann jetzt die Gesamt-Relevanz von $(0.9 + 0.8)/2 = 0.85$ berechnet werden. Trotzdem kann dieser Datensatz noch nicht auf den Ausgabe-Stream geleitet werden, da er unter der Grenz-Relevanz des Ausgabe-Streams $(a + 1)/2 = 0.9$ liegt: Würde z.B. als nächstes von Stream 1 Datensatz a mit einer Relevanz von 0.75 gelesen werden, so hätte dieser eine Gesamt-Relevanz von $(1.0 + 0.75)/2 = 0.875$ und müsste noch vor Datensatz d ausgegeben werden. Bis zu diesem Schritt müssen alle Datensätze im Speicher verbleiben:

Stream 1: ID(Relevanz)

c(0.7)	g(0.6)	i(0.5)	f(0.4)	...

Stream 2: ID(Relevanz)

a(0.7)	f(0.6)	h(0.5)	i(0.4)	...

Puffer für Stream 1: ID(Relevanz)

a(1.0)	e(0.8)

Puffer für Stream 2: ID(Relevanz)

b(1.0)	c(0.9)

Puffer für Ausgabe-Stream: ID(Relevanz)

d(0.85)

Tatsächlich jedoch stellt sich im nächsten Schritt heraus, dass Datensatz a auf Stream 2 nur eine Relevanz von 0.7 hat, so dass dieser auch nur eine Gesamt-Relevanz von $(1.0 + 0.7)/2 = 0.85$ hat. Da die Grenz-Relevanz nun ebenfalls bei $(1.0 + 0.7)/2 = 0.85$ steht, können sowohl Datensatz d als auch Datensatz a (die beide eine mindestens genauso hohe Relevanz haben) auf den Ausgabe-Stream geleitet werden:

Stream 1: ID(Relevanz)

g(0.6)	i(0.5)	f(0.4)	...

Stream 2: ID(Relevanz)

f(0.6)	h(0.5)	i(0.4)	...

Puffer für Stream 1: ID(Relevanz)

e(0.8)

Puffer für Stream 2: ID(Relevanz)

b(1.0)

Puffer für Ausgabe-Stream: ID(Relevanz)

c(0.8)

Ausgabe-Stream: ID(Relevanz)

d(0.85)	a(0.85)	...

Analog arbeitet das Verfahren weiter, wobei sich jedoch schon abzeichnet (vgl. Abbildung 4.13), dass selbst als beide Streams zur Hälfte eingelesen sind, nur 3 Datensätze ausgegeben werden, im Verhältnis zu 18 Datensätzen, die insgesamt (von beiden Streams zusammen) eingelesen wurden.

Die Berechnung von $L_s(N, n)$, $L_r(N, n)$ und $M(N, n)$ auf Basis der Integrale $G(a)$ und $A(a)$ bzw. $A^{-1}(A)$ ist hier relativ einfach und lässt sich anhand von Abb. 4.13 gut veranschaulichen: Definitionsgemäß ist der *Anteil* der ausgegebenen Objekte $A(a) = \frac{n}{N}$. Die Relevanz, bis zu der hin die Streams eingelesen werden müssen, ist demnach $a = A^{-1}(\frac{n}{N})$. Damit ist der *Anteil*

$G(a)$ der einzulesenden Objekte $G(A^{-1}(\frac{n}{N}))$. Die *absolute Zahl* an einzulesenden Objekten $L_s(N,n)$ berechnet sich damit zu:

$$L_s(N,n) = N * G(A^{-1}(\frac{n}{N}))$$

Da der Algorithmus nach Pfeifer/Fuhr keinen random access benötigt, ist $L_r(N,n) = 0$. Die Anzahl zu puffernder Objekte ist gleich der Anzahl gelesener Objekte minus der Anzahl ausgegebener Objekte, und berechnet sich daher zu:

$$M(N,n) = N * G(A^{-1}(\frac{n}{N})) - n$$

Für die verschiedenen Verteilungen ergibt sich konkret:

Gleichverteilung:
$L_s(N,n) = \sqrt{8nN}$
$L_r(N,n) = 0$
$M(N,n) = \sqrt{8nN} - n$

Normalverteilung:
$L_s(N,n) = \sqrt{13.3nN}$
$L_r(N,n) = 0$
$M(N,n) = \sqrt{13.3nN} - n$

Exponentialverteilung:
$L_s(N,n) = \sqrt{42nN}$
$L_r(N,n) = 0$
$M(N,n) = \sqrt{42nN} - n$

3D-Gleichverteilung:
$L_s(N,n) = \sqrt[3]{162N^2n}$
$L_r(N,n) = 0$
$M(N,n) = \sqrt[3]{162N^2n} - n$

Abbildung 4.14 zeigt diese Graphen für $n = 25$ angeforderte Objekte. Für die zweidimensionalen Verteilungen skaliert das Laufzeitverhalten quadratwurzelförmig mit der Stream-Größe N und der Anzahl angeforderter Objekte n, wobei die Unterschiede im Wesentlichen in konstanten Faktoren liegen. Für die dreidimensionale Gleichverteilung skaliert das Verfahren mit $n^{\frac{1}{3}}$, in Abhängigkeit von N jedoch zu $N^{\frac{2}{3}}$. Für noch höherdimensionale Verteilungen, also mehr zu verknüpfende Attribute, nähert sich das Skalierungsverhalten bezüglich N mit $L_s \sim N^{\frac{dim-1}{dim}}$ immer mehr der Linearität an.

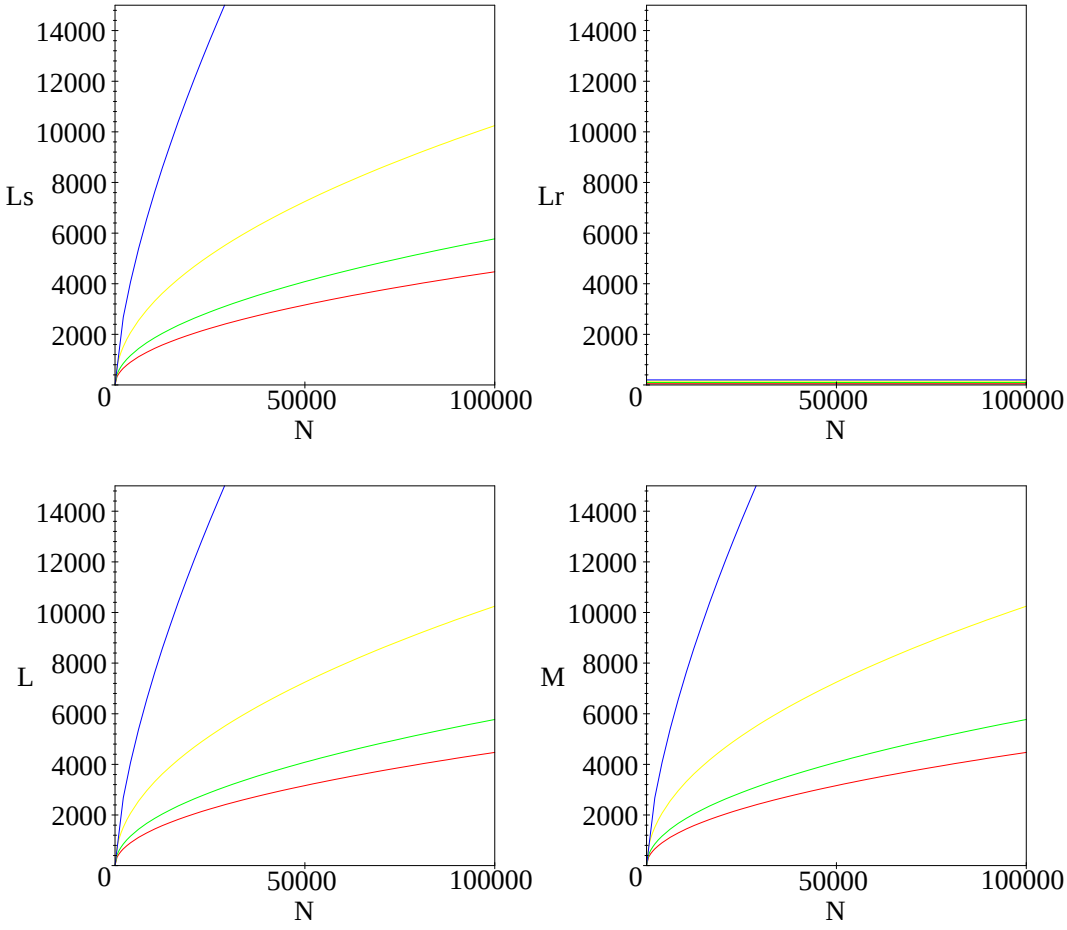


Abbildung 4.14: Pfeifer-Fuhr-Algorithmus: sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) sowie Gesamtlaufzeitkosten $L(N,n)$ (unten links, exemplarisch für $h = 3$) und Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$.

4.3.4 Fagin-Algorithmus (FA)

Fagin entwickelt in [19] einen Algorithmus zur Verknüpfung von mehreren Fuzzy-Streams, der die Voraussetzung ausnutzt, dass auf vielen Datenbanken ein random access möglich ist und dadurch die Anzahl an sorted accesses deutlich reduziert werden kann.

Das Prinzip des Algorithmus ist das folgende: Wenn n Datensätze vom Benutzer angefordert werden, so liest der Algorithmus von allen Eingangs-Streams solange Datensätze ein, bis es zu n matches (d.h. Vorkommen eines Datensatzes in *allen* Streams) kommt. Dadurch ist sichergestellt, dass die n Ergebnisse mit der höchsten Gesamt-Relevanz zu diesem Zeitpunkt von mindestens einem der Streams eingelesen wurden und es muss nun für alle bisher eingelesenen Datensätze ein random access auf den jeweils anderen Streams stattfinden (für bestimmte Aggregations-Operatoren beweist Fagin, dass es genügt, die random accesses nur für Datensätze von einem bestimmten Stream durchzuführen, dazu später mehr). Für die eingelesenen Datensätze kann nun die endgültige Relevanz für den Ausgangs-Stream berechnet werden, und es können die gesuchten n Datensätze ausgegeben werden.

Als Beispiel seien die folgenden Streams gegeben, und es seien $n = 2$ Datensätze vom Benutzer angefordert. Als Aggregations-Operator diene zunächst die Minimums-Norm. Die Verteilung der Datensätze ist in Abbildung 4.15 visualisiert:

Stream 1: ID(Relevanz)

b(1.0)	a(0.9)	e(0.9)	c(0.7)	d(0.6)	...	f(0.1)	...

Stream 2: ID(Relevanz)

d(0.9)	a(0.9)	c(0.7)	f(0.7)	g(0.7)	b(0.6)	e(0.5)	...

Nachdem von jeweils beiden Streams die ersten vier Datensätze eingelesen wurden, sind die geforderten zwei matches vorhanden (a und c). Für sie können bereits die endgültigen Relevanzen für den Ausgangs-Stream berechnet werden ($\mu(a) = min(0.90, 0.90) = 0.90$ und $\mu(c) = min(0.70, 0.70) = 0.70$). Für die Minimums-Norm als Aggregations-Operator beweist Fagin, dass der random access nicht für alle bisher gelesenen Datensätze nötig ist, sondern lediglich für alle Datensätze, die auf dem Stream gelesen wurden, auf dem der letzte match (hier c) mit der kleinsten Relevanz gelesen wurde, also in diesem Fall Stream 1 ($\mu_1(c) = 0.7$). In diesem Fall müssen also für die Datensätze d und f random accesses auf Stream 1 stattfinden, die eine Relevanz von $\mu_1(d) = 0.60$ und $\mu_1(f) = 0.1$ liefern. Die Gesamt-Relevanzen für d und f berechnen sich also als $\mu(d) = min(0.6, 0.9) = 0.6$ und $\mu(f) = min(0.1, 0.7) = 0.1$. Damit ergibt sich für die höchsten $n = 2$ Datensätze, die auf den Ausgangs-Stream geleitet werden können, das Ergebnis:

Ausgangs-Stream: ID(Relevanz)

a(0.90)	c(0.70)	...

In diesem Beispiel ist auch die eingangs nur angerissene Tatsache ersichtlich, inwiefern der Algorithmus von der Wahl des Aggregations-Operators abhängt: Wählt man als Aggre-

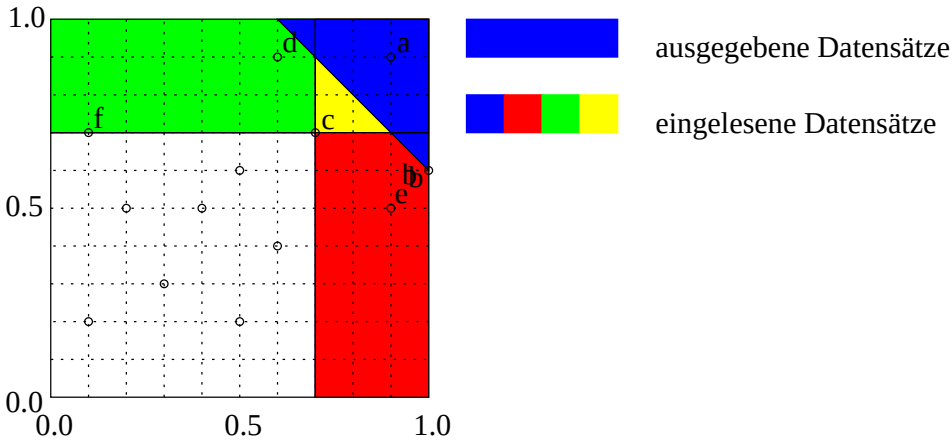


Abbildung 4.15: Verteilung von Datensätzen über 2 gewichtete Streams. Zustandsaufnahme für Grenz-Relevanz $\mu = 0.7$.

gations-Operator z.B. das arithmetische Mittel, so muss für *alle* bisher eingelesenen Datensätze ein random access erfolgen.[2] So ergäben sich in diesem Fall die Relevanzen $\mu(a) = \frac{0.9+0.9}{2} = 0.9$, $\mu(b) = \frac{1.0+0.6}{2} = 0.8$, $\mu(c) = \frac{0.7+0.7}{2} = 0.7$, $\mu(d) = \frac{0.6+0.9}{2} = 0.75$, $\mu(e) = \frac{0.9+0.5}{2} = 0.7$, $\mu(f) = \frac{0.1+0.7}{2} = 0.4$ und es müssten statt a und c die Datensätze a und b ausgegeben werden.

Genaugenommen ist dieser Algorithmus nicht als Streaming-Algorithmus definiert, stattdessen wird die Anzahl an gewünschten Ergebnissen n als Eingabe benötigt - die Ausweitung auf einen echten Streaming-Algorithmus ist jedoch naheliegend. Für zwei Eingangs-Streams lautet der entsprechende Pseudocode:

```
INPUT: RandomAccessStream s1, s2; Integer n;
OUTPUT: Stream out;

SortedList s1Buffer, s2Buffer, outBuffer;
Integer i, matches=0;
Dataset d1, d2;

// Sequentieller Zugriff bis zu n matches:

WHILE matches<n DO

  IF s1.currentRelevance>s2.currentRelevance THEN
    d1:=s1.read();
```

[2] Fagin untersucht und beweist die Kriterien für die konkrete Wahl des Algorithmus in Abhängigkeit der Eigenschaften des Operators in [20].

```
    s1Buffer.add(d1);
    IF s2Buffer.contains(d1) THEN
      matches:=matches+1;
    END IF
  ELSE
    d2:=s2.read();
    s2Buffer.add(d2);
    IF s1Buffer.contains(d2) THEN
      matches:=matches+1;
    END IF
  END IF

END WHILE

// random access und Berechnung der Relevanzen:

// ...zunaechst fuer die matches:

FOR EACH d1 IN s1Buffer AND d2 IN s2Buffer WITH d1.id==d2.id DO
  d1.relevance=agg(d1.relevance,d2.relevance)
  outBuffer.add(d1);
  s1Buffer.remove(d1);
  s2Buffer.remove(d2);
END FOR

// ...dann fuer die verbleibenden Datensaetze:

FOR EACH d1 IN s1Buffer DO
  d1.relevance=agg(d1.relevance,s2.randomAccess(d1));
  s1Buffer.remove(d1);
  outBuffer.add(d1);
END FOR

FOR EACH d2 IN s2Buffer DO
  d2.relevance=agg(s1.randomAccess(d2),d2.relevance);
  s2Buffer.remove(d2);
  outBuffer.add(d2);
END FOR

// n Elemente auf Output-Stream schreiben:

FOR i:=0 TO n-1 DO
  out.write(outBuffer.elementAt(n));
END FOR
```

Die Abschätzung der Laufzeitkosten ergibt sich wie folgt: Definitionsgemäß werden soviele Objekte eingelesen, bis die Anzahl der matches n beträgt, also bis zu dem Punkt $M(a) = \frac{n}{N} \Leftrightarrow a = M^{-1}(\frac{n}{N})$. Der Anteil eingelesener Objekte $G(a)$ entspricht dann $G(M^{-1}(\frac{n}{N}))$, die absolute Zahl demnach

$$L_s(N,n) = N * G(M^{-1}(\frac{n}{N}))$$

Weiterhin müssen für alle diese Objekte - außer für die n matches - random accesses auf allen $(dim-1)$ anderen Streams, auf denen das jeweilige Objekt noch nicht gelesen wurde, stattfinden:

$$L_r(N,n) \quad = \quad (dim-1) * (L_s(N,n) - n)$$

Zwischengespeichert werden müssen wieder alle Objekte, die nicht ausgegeben werden:

$$M(N,n) \quad = \quad L_s(N,n) - n$$

Für die verschiedenen Verteilungen ergibt sich konkret:

Gleichverteilung:
$L_s(N,n) = \sqrt{4nN}$
$L_r(N,n) = \sqrt{4nN} - n$
$M(N,n) = \sqrt{4nN} - n$

Normalverteilung:
$L_s(N,n) = \sqrt{6.5nN}$
$L_r(N,n) = \sqrt{6.5nN} - n$
$M(N,n) = \sqrt{6.5nN} - n$

Exponentialverteilung:
$L_s(N,n) = \sqrt{21nN}$
$L_r(N,n) = \sqrt{21nN} - n$
$M(N,n) = \sqrt{21nN} - n$

3D-Gleichverteilung:
$L_s(N,n) = \sqrt[3]{27nN^2}$
$L_r(N,n) = \sqrt[3]{216nN^2} - 2n$
$M(N,n) = \sqrt[3]{27nN^2} - n$

Abbildung 4.16 zeigt diese Graphen. Das generelle Skalierungsverhalten ist dabei dasselbe wie das des Pfeifer-Fuhr-Algorithmus, jedoch sind die konstanten Faktoren um das 1.5- bis 2-fache kleiner. Die prinzipiell genauso vielen nötigen random accesses, die beim Pfeifer-Fuhr-Algorithmus nicht erforderlich sind, wiegen dies in der Praxis jedoch meistens auf.

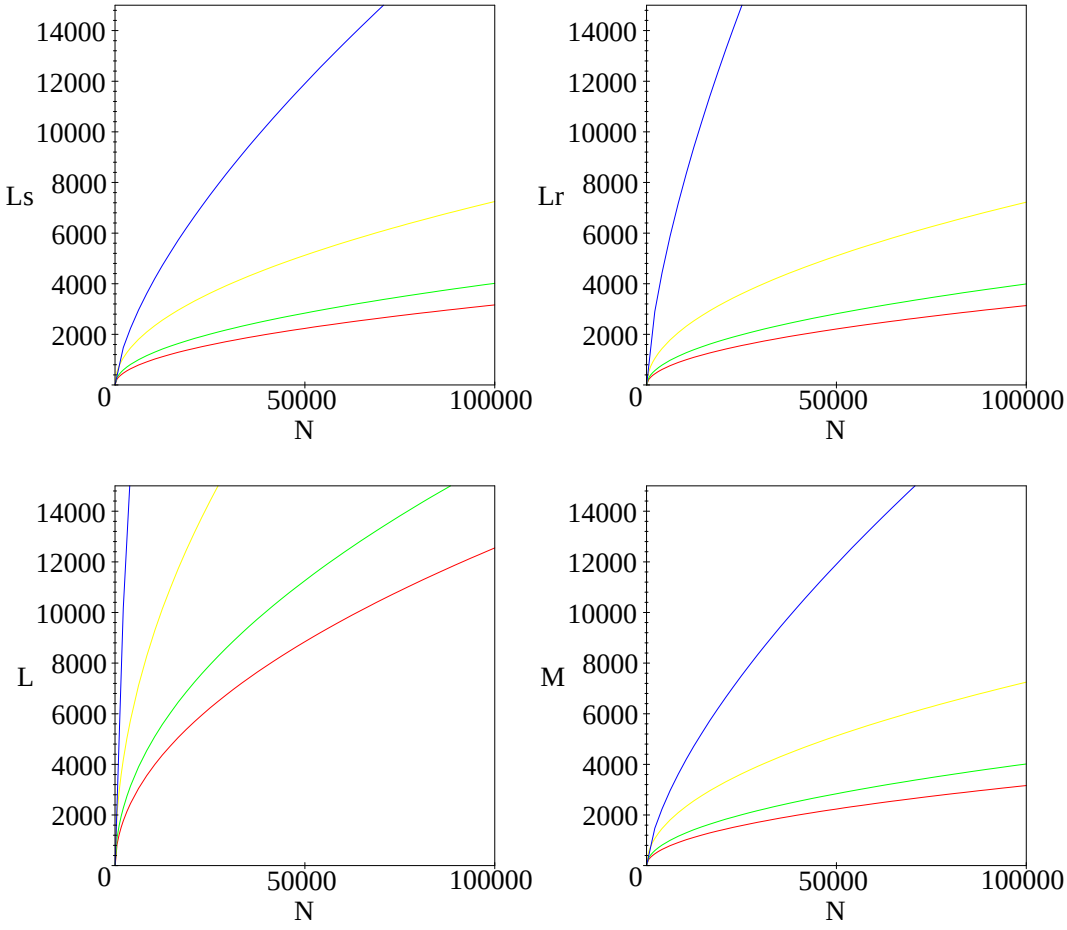


Abbildung 4.16: Fagin-Algorithmus: sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) sowie Gesamtlaufzeitkosten $L(N,n)$ (unten links, exemplarisch für $h = 3$) und Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$.

Die entscheidenden Nachteile des Fagin-Algorithmus sind also insbesondere der hohe Speicherbedarf und die teuren random accesses. In [21] werden drei Algorithmen entwickelt, die diese Probleme umgehen. Der Threshold-Algorithmus (TA) kommt mit konstantem Speicher aus. Der No-Random-Access-Algorithmus (NRA) verzichtet vollständig auf random accesses, wobei er jedoch nur die Rangfolge der Ergebnisse, nicht aber ihre genauen Relevanzen berechnet, und dadurch weitere Zugriffe einsparen kann. Der Combined-Algorithmus (CA) optimiert ausgehend vom Verhältnis der Kosten für einen sorted access zu einem random access die Anzahl der nötigen Zugriffe.

4.3.5 Threshold-Algorithmus (TA)

Der in [21] entwickelte Threshold-Algorithmus zur Verknüpfung von Fuzzy-Streams kommt im Gegensatz zu den bisher vorgestellten Algorithmen mit minimalem Speicher aus - insbesondere ist der benötigte Speicherplatz unabhängig von der Größe der Streams N. Erkauft wird dieser Vorteil durch eine erhöhte Anzahl an random accesses.

Beim Threshold-Algorithmus findet zu jedem mittels sorted access eingelesenen Objekt unmittelbar ein random access auf das bzw. die anderen Attribute statt, so dass die endgültige Relevanz sofort berechnet werden kann. Liegt diese Relevanz über dem sog. threshold-value, der Grenz-Relevanz, welche sich aus der Aggregation der aktuellen Zugehörigkeitsgrade der Streams berechnet, und sind mindestens n derartige Objekte eingelesen worden, so können eben diese n Objekte mit den höchsten Relevanzen ausgegeben werden. Der Puffer, in dem die Datensätze zwischengespeichert werden, muss dabei immer nur die n Datensätze mit der höchsten Relevanz enthalten.

Der Algorithmus ist genau wie der Fagin-Algorithmus nicht als Streaming-Algorithmus definiert und benötigt ebenfalls n als externen Parameter. Eine Ausweitung auf echtes Streaming ist im Gegensatz zum Fagin-Algorithmus nicht trivial, da von vorneherein ein n Datensätze großer Puffer benötigt wird, wobei n im Voraus nicht bekannt und insbesondere nicht nach oben beschränkt ist. Man kann nun entweder n künstlich nach oben beschränken, dem Benutzer also eine Maximalanzahl an Datensätzen präsentieren oder aber den Puffer nicht beschränken, sondern mitwachsen lassen (wobei dann aber *jeder eingelesene* Datensatz gepuffert werden muss, das Wachstum also nicht mehr linear zu n erfolgt, sondern einen erheblich höheren Aufwand nach sich zieht, wie im Folgenden gezeigt wird). Formal wird der Algorithmus (ohne diese Modifikationen) wie folgt implementiert:

```
INPUT: RandomAccessStream s1, s2; Integer n;
OUTPUT: Stream out;

SortedSet outBuffer; // sortiert nach Relevanz
Dataset d1,d2;
Double r1,r2,thresholdValue:=1.0;

// sequentieller und random access bis zu n validen Datensaetzen:
```

```
WHILE (outBuffer.size()<n) OR
      (outBuffer.elementAt(n).relevance<thresholdValue) DO

  IF s1.currentRelevance>s2.currentRelevance THEN
    d1:=s1.read();
    r1:=d1.relevance;
    d1.relevance:=agg(d1.relevance,s2.randomAccess(d1));
    outBuffer.add(d1);
  ELSE
    d2:=s2.read();
    r2:=d2.relevance;
    d2.relevance:=agg(d2.relevance,s1.randomAccess(d2));
    outBuffer.add(d2);
  END IF

  // Groessenbeschraenkung des Puffers:
  IF outBuffer.size()>n DO
    outBuffer.removeLast();
  END IF

  thresholdValue:=agg(r1,r2);

END WHILE

// erste n Elemente auf Output-Stream schreiben:

FOR i FROM 0 TO n-1 DO
  out.write(outBuffer.elementAt(i));
END FOR
```

Zur Ausführung ein Beispiel der in Abbildung 4.17 gezeigten Verteilung:

Stream 1: ID(Relevanz)

a(0.9)	c(0.9)	e(0.7)	b(0.7)	...	d(0.1)	...

Stream 2: ID(Relevanz)

a(0.9)	b(0.8)	d(0.7)	...	c(0.5)	...	e(0.1)	...

Seien vom Benutzer $n = 3$ Datensätze angefordert. Zunächst wird sowohl von Stream 1, als auch von Stream 2 Datensatz a gelesen, der jeweils eine Relevanz von 0.9 und somit auch eine Gesamt-Relevanz von 0.9 hat. Danach wird von Stream 1 Datensatz c mit einer Relevanz von 0.9 und von Stream 2 Datensatz b mit einer Relevanz von 0.8 gelesen. Durch

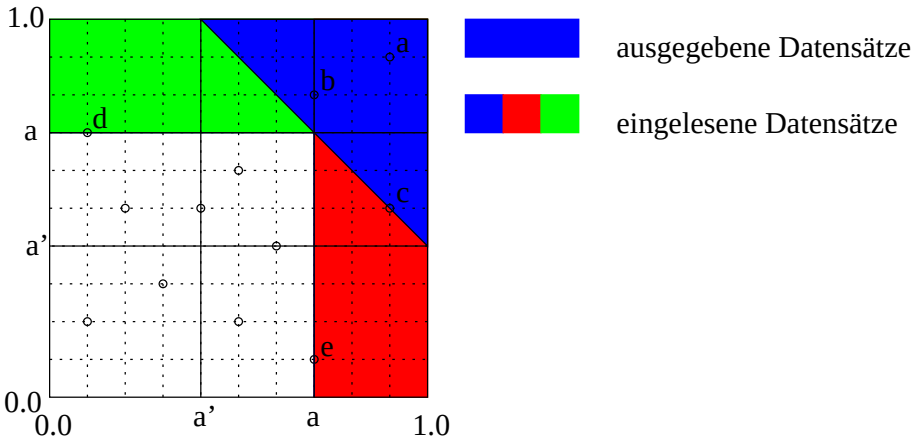


Abbildung 4.17: Verteilung von Datensätzen über zwei Attribute. Markierte Integrale $G(a)$ und $A(a)$ für den Threshold-Algorithmus.

die random accesses können unmittelbar die Gesamt-Relevanzen berechnet werden und die $n = 3$ Datensätze mit den höchsten Relevanzen im Ausgabepuffer gehalten werden:

Stream 1: ID(Relevanz)

e(0.7)	b(0.7)	...	d(0.1)	...

Stream 2: ID(Relevanz)

d(0.7)	...	c(0.5)	...	e(0.1)	...

Ausgabepuffer: ID(Relevanz)

a(0.9)	b(0.75)	c(0.7)

Da der threshold-value zu diesem Zeitpunkt bei $\frac{0.9+0.8}{2} = 0.85$ und noch nicht $n = 3$ Datensätze über diesem Wert liegen, fährt der Algorithmus fort und liest im folgenden Schritt von Stream 1 Datensatz e und von Stream 2 Datensatz d. Durch die random accesses ergeben sich wieder unmittelbar die Gesamt-Relevanzen von $\frac{0.7+0.1}{2} = 0.4$ für e und ebenfalls $\frac{0.7+0.1}{2} = 0.4$ für d. Obwohl diese Relevanzen unter dem threshold-value liegen, stoppt der Algorithmus an dieser Stelle, da der threshold-value jetzt auf $\frac{0.7+0.7}{2} = 0.7$ gefallen ist, und damit $n = 3$ Datensätze (a, b und c) vorliegen, die eine mindestens genauso hohe Relevanz haben. Diese Datensätze können daher ausgegeben werden.

Die Kosten des Threshold-Algorithmus berechnen sich wie folgt, wie man sich anhand von Abbildung 4.17 leicht verdeutlichen kann: Die Anzahl gelesener Datensätze beträgt wie üblich $L_s(N, n) = N * G(a)$. Der Anteil ausgegebener Datensätze beträgt zu diesem Zeitpunkt $A(a') = \frac{n}{N}$, diese sind in ihren Relevanzen durch die Äquirelevanz-Linie $\frac{a+a}{2} = \frac{a'+1}{2}$ beschränkt (analog bei 3 Streams durch die Äquirelevanz-Ebene $\frac{a'+1+1}{3} = \frac{a+a+a}{3}$):

$$
\begin{aligned}
L_s(N,n) &= N * G(a) \\
&\quad mit\ a' = A^{-1}(\frac{n}{N}) \\
&\quad mit\ a = \frac{a'+1}{2}\ fuer\ dim = 2 \\
&\quad mit\ a = \frac{a'+2}{3}\ fuer\ dim = 3
\end{aligned}
$$

Random accesses finden für jedes der eingelesenen Objekte auf jedem verbleibenden Stream statt:

$$L_r(N,n) = (dim-1) * L_s(N,n)$$

Damit erhält man konkret:

Gleichverteilung:
$L_s(N,n) = L_r(N,n) = \sqrt{2Nn}$
$M(N,n) = n$

Normalverteilung:
$L_s(N,n) = L_r(N,n) = \sqrt{3.3Nn}$
$M(N,n) = n$

Exponentialverteilung:
$L_s(N,n) = L_r(N,n) = \sqrt{10.5nN}$
$M(N,n) = n$

3D-Gleichverteilung:
$L_s(N,n) = \sqrt[3]{6N^2n}$
$L_r(N,n) = \sqrt[3]{48N^2n}$
$M(N,n) = n$

Die Graphen dieser Kostenfunktionen sind in Abbildung 4.18 zu sehen. Die Graphen der zweidimensionalen Verteilungen weisen eine sowohl von der Anzahl angeforderter Datensätze n als auch von der Größe der Streams N quadratwurzelförmige Abhängigkeit auf. Für mehr zu verknüpfende Streams ist die Kostenfunktion wieder proportional zu $N^{\frac{dim-1}{dim}}$. Die konstanten Skalierungsfaktoren liegen dabei deutlich unter denen des Fagin-Algorithmus, und darüber hinaus ist der Speicherverbrauch im Gegensatz zum Fagin-Algorithmus konstant.

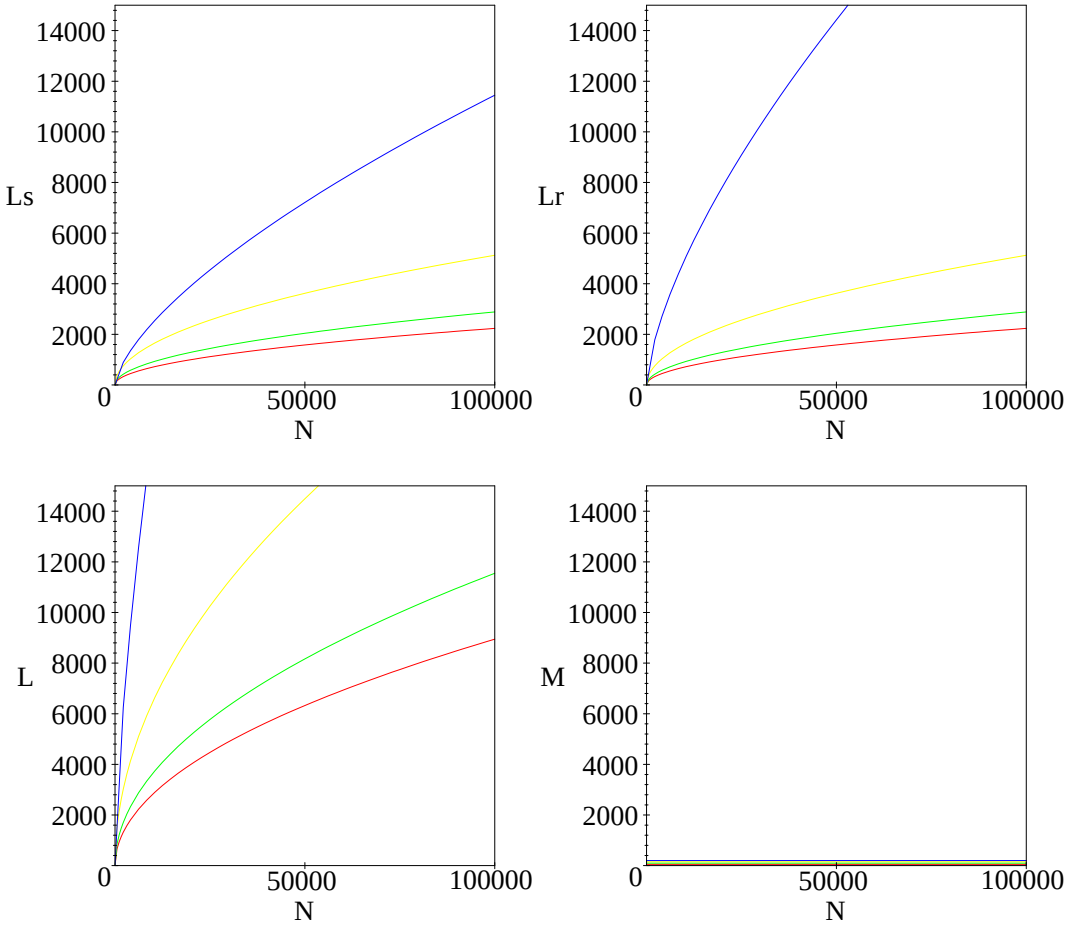


Abbildung 4.18: Threshold-Algorithmus: sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) sowie Gesamtlaufzeitkosten $L(N,n)$ (unten links, exemplarisch für $h = 3$) und Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$.

4.3.6 No-Random-Access-Algorithmus (NRA)

Genau wie der Algorithmus nach Pfeifer/Fuhr kommt der in [21] entwickelte NRA-Algorithmus ohne random accesses aus und verknüpft zwei oder mehr Fuzzy-Streams alleine durch sequentiellen Zugriff. Dieser Algorithmus versucht, auch die Anzahl an sorted accesses auf das minimal mögliche Maß zu reduzieren. Dabei nutzt er die Tatsache aus, dass oft gar nicht die konkrete Relevanz eines Datensatzes interessant ist, sondern vielmehr die Reihenfolge innerhalb der Ergebnismenge das entscheidende Kriterium darstellt. Kann zu einem bestimmten Zeitpunkt also die Relevanz eines Datensatzes nach unten hin durch eine Abschätzung begrenzt werden und gleichzeitig sichergestellt werden, dass kein anderer Datensatz mehr eine höhere Relevanz bekommen kann, so kann der betreffende Datensatz - ohne seine exakte Relevanz zu kennen - auf den Ausgabe-Stream geleitet werden.

Dazu ein Beispiel der in Abb. 4.19 gezeigten Verteilung. Gegeben seien die folgenden Eingangs-Streams:

Stream 1: ID(Relevanz)

a(0.9)	c(0.5)	d(0.3)	b(0.2)	...

Stream 2: ID(Relevanz)

b(1.0)	a(0.8)	d(0.3)	c(0.2)	...

Im ersten Schritt wird von Stream 1 der Datensatz a mit einer Relevanz von 0.9 gelesen und von Stream 2 Datensatz b mit einer Relevanz von 1.0. Noch kann kein Ergebnis ausgegeben werden.

Im nächsten Schritt wird von Stream 1 Datensatz c mit einer Relevanz von 0.5 gelesen und von Stream 2 Datensatz a mit einer Relevanz von 0.8. Damit kann die Gesamt-Relevanz für a als $\frac{0.9+0.8}{2} = 0.85$ berechnet werden. Zu diesem Zeitpunkt kann Datensatz b eine Gesamt-Relevanz von maximal $\frac{1.0+0.5}{2} = 0.75$ bekommen (falls er auf Stream 1, auf dem er noch nicht gelesen wurde, eine Relevanz von 0.5 hätte, was zu diesem Zeitpunkt noch nicht klar ist) und Datensatz c eine maximale Gesamt-Relevanz von $\frac{0.5+0.8}{2} = 0.65$ (falls er auf Stream 2 eine Relevanz von 0.8 hätte). Datensätze, die auf noch keinem Stream gelesen wurden, können ebenfalls maximal eine Gesamt-Relevanz von $\frac{0.5+0.8}{2} = 0.65$ erhalten. Da Datensatz a über allen diesen Relevanzen liegt, kann er als erster auf den Ergebnis-Stream geleitet werden:

Stream 1: ID(Relevanz)

d(0.3)	b(0.2)	...

Stream 2: ID(Relevanz)

d(0.3)	c(0.2)	...

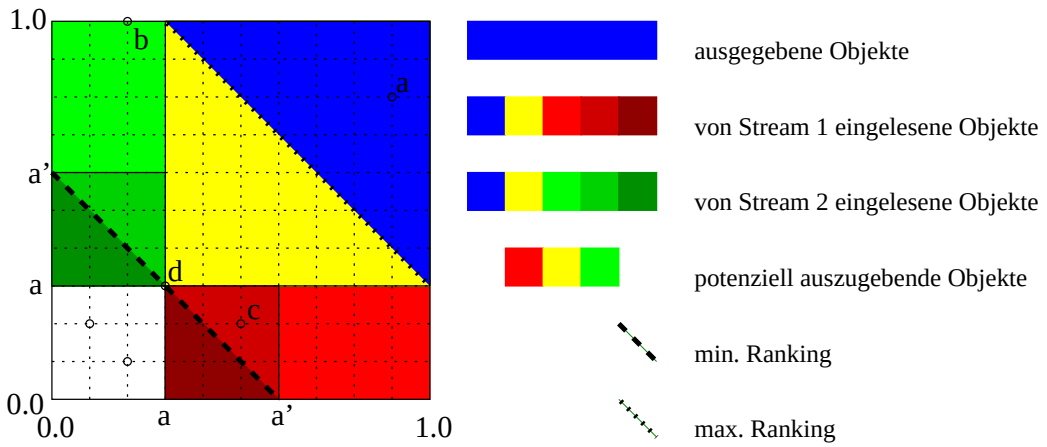


Abbildung 4.19: Verteilung von Datensätzen über zwei Attribute, markierte Integrale für den NRA-Algorithmus.

Ausgabepuffer:

b	c
μ_1 =?	$\mu_1 = 0.5$
$\mu_2 = 1.0$	μ_2 =?
$\mu_{best} = 0.75$	$\mu_{best} = 0.65$
$\mu_{worst} = 0.5$	$\mu_{worst} = 0.25$

Ausgabe-Stream:

a(0.85)	...

Im dritten Schritt wird von Stream 1 Datensatz d mit einer Relevanz von 0.3 gelesen und von Stream 2 ebenfalls Datensatz d mit einer Relevanz von 0.3. Datensatz b hat bekanntermaßen mindestens eine Gesamt-Relevanz von $\frac{1.0+0.0}{2} = 0.5$. Die „best relevance" von Datensatz c berechnet sich nun zu $\frac{0.5+0.3}{2} = 0.4$, die von Datensatz d zu $\frac{0.3+0.3}{2} = 0.3$, ebenso wie alle noch von keinem Stream gelesenen Datensätze maximal eine Gesamt-Relevanz von $\frac{0.3+0.3}{2} = 0.3$ haben können. Damit ist sichergestellt, dass b die (nach dem bereits ausgegebenen a) nächsthöchste Relevanz hat und damit ausgegeben werden kann:

Stream 1: ID(Relevanz)

b(0.2)	...

Stream 2: ID(Relevanz)

c(0.2)	...

Ausgabepuffer:

c	d
$\mu_1 = 0.5$	$\mu_1 = 0.3$
$\mu_2 =?$	$\mu_2 = 0.3$
$\mu_{best} = 0.4$	$\mu_{best} = 0.3$
$\mu_{worst} = 0.25$	$\mu_{worst} = 0.3$

Ausgabe-Stream: ID(Relevanz)

a(0.85)	b(0.5..0.65)	...

Der entscheidende Aspekt, der den NRA-Algorithmus von den bisher vorgestellten Algorithmen unterscheidet ist, dass diese Aussage getroffen und b ausgegeben werden kann, obwohl die genaue Gesamt-Relevanz von b noch nicht berechnet werden kann.

Der kalkulatorische Aufwand für diese Abschätzung ist jedoch sehr hoch: Wenn für jeden der zu einem bestimmten Zeitpunkt eingelesenen i Datensätze in jedem der i Schritte die obere und untere Grenze neu berechnet werden, so steigt der Aufwand mit $O(i^2)$ an. Für (durchaus realistische) Werte für i von z.B. 10.000 müssten in jedem Schritt 10.000, insgesamt also $100.000.000$ obere und untere Schranken berechnet werden sowie Minimal- und Maximalwerte ermittelt werden. Dieser Aufwand wird in der Praxis dadurch reduziert, dass die Schranken nur alle d Schritte neu berechnet werden. (Da sich normalerweise die Schranken in jedem Schritt nur geringfügig ändern, wird das Skalierungsverhalten bezüglich der Datenbankzugriffe dadurch kaum negativ beeinflusst. Dieser Aspekt wird daher bei der Aufwandsabschätzung vernachlässigt, und es wird zugunsten das Algorithmus davon ausgegangen, dass die Berechnung *mit minimalem kalkulatorischen Aufwand* in *jedem* Schritt durchgeführt wird.)

Genaugenommen wird auch der NRA nicht als Streaming-Algorithmus definiert und benötigt n als zusätzlichen Parameter, die Ausweitung auf echtes Streaming ist aber naheliegend. Formal arbeitet der Algorithmus wie folgt:

```
INPUT: Stream s1, s2; Integer d;
OUTPUT: Stream out;

SortedList s1Buffer,s2Buffer,outBuffer;
Dataset d1,d2,maxWorstD,maxBestD,d;
Integer counter;
Double r1,r2;

WHILE NOT out.closed() DO

  // naechsten (noch nicht vom anderen Stream
  // gelesenen) Datensatz lesen:

  IF s1.currentRelevance>s2.currentRelevance THEN
    DO
```

```
    d1:=s1.read();
  WHILE outBuffer.contains(d1.id);
  r1:=d1.relevance;
  s1Buffer.add(d1);
ELSE
  DO
    d2:=s2.read();
  WHILE outBuffer.contains(d2.id);
  r2:=d2.relevance;
  s2Buffer.add(d2);
END IF

// worst- und best-relevances berechnen (jeden d-ten Schritt):

counter:=counter+1;

IF counter\%d==0 THEN

  FOR EACH d1 IN s1Buffer DO
    IF s2Buffer.contains(d1.id) THEN
      d2:=s2Buffer.getElement(d1.id);
      d1.worstRelevance:=d1.bestRelevance:=
        agg(d1.relevance,d2.relevance);
    ELSE
      d1.worstRelevance:=agg(d1.relevance,0);
      d1.bestRelevance:=agg(d1.relevance,r2);
    END IF
  END FOR

  FOR EACH d2 IN s2Buffer DO
    IF s1Buffer.contains(d2.id) THEN
      d1:=s1Buffer.getElement(d2.id);
      d2.worstRelevance:=d2.bestRelevance:=
        agg(d1.relevance,d2.relevance);
    ELSE
      d2.worstRelevance:=agg(d2.relevance,0);
      d2.bestRelevance:=agg(d2.relevance,r1);
    END IF
  END FOR

END IF

// Datensatz mit maximaler worst-relevance suchen:

FOR EACH d IN s1Buffer OR IN s2Buffer DO
```

```
    IF (d.worstRelevance>maxWorstD.worstRelevance
        OR (d.worstRelevance==maxWorstD.worstRelevance AND
            d.bestRelevance>maxWorstD.bestRelevance) ) THEN
      maxWorstD:=d;
    END IF
  END FOR

  // Datensatz mit maximaler best-relevance suchen:

  FOR EACH d IN s1Buffer OR s2Buffer DO
    IF (d.id!=maxWorstD.id AND
        d.bestRelevance>maxBestD.bestRelevance) THEN
      maxBestD:=d;
    END IF
  END FOR

  // ersten Datensatz wenn moeglich auf
  // Outputstream schreiben:

  IF maxWorstD.worstRelevance>=maxBestD.bestRelevance THEN
    outBuffer.add(maxWorstD);
    s1Buffer.remove(maxWorstD);
    s2Buffer.remove(maxWorstD);
    out.write(maxWorstD);
  END IF

END WHILE
```

Zur Abschätzung der Kostenmaße ist die Eingrenzung der „worst relevance", welche ein Datensatz erhalten kann, entscheidend: Die maximale worst relevance ergibt sich, wenn der Datensatz auf einem der Streams mit einer Relevanz von 1.0 eingelesen wurde. Es beträgt dann $\frac{1.0+0.0}{2} = 0.5$. Die maximale „best relevance", welche ein noch nicht eingelesener Datensatz haben kann, beträgt $\frac{a+a}{2} = a$. Demnach kann für $a > 0.5$, also bei noch nicht bis zur Hälfte eingelesenen Streams, kein Datensatz ausgegeben werden, der nicht bereits von beiden Streams gelesen wurde (womit sich seine exakte Relevanz - welche dann über 0.5 liegt - berechnen ließe.)

Dies ist beim Vergleich der Abbildungen 4.19 und 4.20 gut zu erkennen, in denen der hellrote und der hellgrüne Bereich der potentiell auszugebenden Datensätze nur dann existieren, wenn $a < 0.5$ ist. Für $a > 0.5$ gehören lediglich die Datensätze im gelb gekennzeichneten Bereich zu den potentiell auszugebenden. Dieser Fall ist auch der entscheidende Bereich, da die Streams nur dann bis zu einer Relevanz von weniger als 0.5 eingelesen werden müssen, wenn sie entweder sehr klein sind (wobei die Performance dann ohnehin unkritisch ist) oder einer eher pathologischen Verteilung unterliegen.

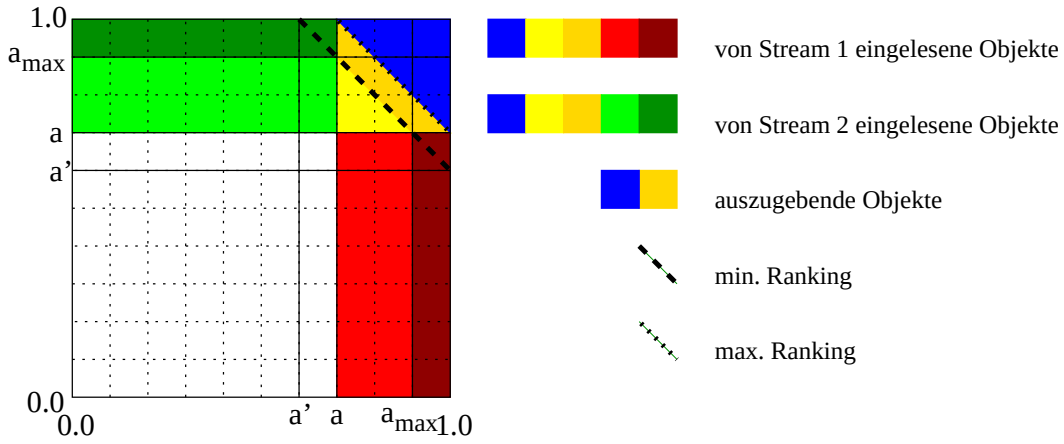


Abbildung 4.20: Verteilung von Datensätzen über zwei Attribute, markierte Integrale für den NRA-Algorithmus.

Trotzdem können auch für $a > 0.5$ mehr Datensätze ausgegeben werden, als beim Pfeifer-Fuhr-Algorithmus: Sofern keine Datensätze existieren, die auf einem der Streams eine Relevanz von mehr als a_{max} haben (also keine Datensätze im dunkelgrün bzw. dunkelrot gekennzeichneten Bereich liegen), ist sichergestellt, dass die im orange gekennzeichneten Bereich liegenden Datensätze die maximal mögliche Gesamt-Relevanz besitzen und ausgegeben werden können. (Je größer die Streams, desto geringer ist allerdings die Wahrscheinlichkeit, dass im dunkelgrün bzw. dunkelrot gekennzeichneten Bereich keine weiteren Datensätze liegen, so dass mit steigendem N im Wesentlichen der blau gekennzeichnete Bereich, der dem des Algorithmus nach Pfeifer/Fuhr entspricht, relevant ist.)

a_{max} ist nach unten durch a und nach oben durch 1.0 beschränkt. Damit ist die erwartete Anzahl der sorted accesses des NRA-Algorithmus nach unten durch die Anzahl der sorted accesses des Threshold-Algorithmus (vgl. Kapitel 4.3.5) beschränkt ($\underline{L_s}(N,n)$) und nach oben durch die Anzahl an sorted accesses des Algorithmus nach Pfeifer/Fuhr (vgl. Kapitel 4.3.3, $\overline{L_s}(N,n)$). Die Anzahl an random accesses beträgt 0 und die Schranken für den zu erwartenden Speicherverbrauch $M(N,n)$ berechnen sich wieder durch die Differenz der eingelesenen und ausgegeben Datensätze:

$$
\begin{aligned}
\overline{L_s}(N,n) &= N * G(A^{-1}(\frac{n}{N})) \\
\underline{L_s}(N,n) &= N * G(a) \\
L_r(N,n) &= 0 \\
\overline{M}(N,n) &= N * G(A^{-1}(\frac{n}{N})) - n \\
\underline{M}(N,n) &= N * G(a) - n
\end{aligned}
$$

$$mit\ a' = A^{-1}(\frac{n}{N})$$
$$mit\ \frac{a'+1}{2} = \frac{a+a}{2} \Leftrightarrow a = \frac{a'+1}{2}\ fuer\ dim = 2$$
$$mit\ \frac{a'+1+1}{3} = \frac{a+a+a}{3} \Leftrightarrow a = \frac{a'+2}{3}\ fuer\ dim = 3$$

Konkret ergibt sich für die verschiedenen Verteilungen:

Gleichverteilung:
$\underline{L_s}(N,n) = \sqrt{2Nn}$ $\quad$ $\overline{L_s}(N,n) = \sqrt{8Nn}$
$\overline{L_r}(N,n) = 0$
$\underline{M}(N,n) = \sqrt{2Nn} - n$ $\quad$ $\overline{M}(N,n) = \sqrt{8Nn} - n$

Normalverteilung:
$\underline{L_s}(N,n) = \sqrt{3.3Nn}$ $\quad$ $\overline{L_s}(N,n) = \sqrt{13.3Nn}$
$\overline{L_r}(N,n) = 0$
$\underline{M}(N,n) = \sqrt{3.3Nn} - n$ $\quad$ $\overline{M}(N,n) = \sqrt{13.3Nn} - n$

Exponentialverteilung:
$\underline{L_s}(N,n) = \sqrt{10.5nN}$ $\quad$ $\overline{L_s}(N,n) = \sqrt{42nN}$
$\overline{L_r}(N,n) = 0$
$\underline{M}(N,n) = \sqrt{10.5nN} - n$ $\quad$ $\overline{M}(N,n) = \sqrt{42nN} - n$

3D-Gleichverteilung:
$\underline{L_s}(N,n) = \sqrt[3]{6N^2n}$ $\quad$ $\overline{L_s}(N,n) = \sqrt[3]{162N^2n}$
$\overline{L_r}(N,n) = 0$
$\underline{M}(N,n) = \sqrt[3]{6N^2n} - n$ $\quad$ $\overline{M}(N,n) = \sqrt[3]{162N^2n} - n$

Mit dem NRA-Algorithmus steht ein Verfahren zur Verfügung, welches mit minimalen Datenbank-Kosten mehrere Fuzzy-Streams, auf denen kein random access möglich ist, miteinander verknüpft. Dies wird durch einen stark erhöhten kalkulatorischen Aufwand und insbesondere durch lediglich Sicherstellen der richtigen Rangfolge anstelle der Berechnung der konkreten Relevanzen erkauft. Außerdem unterliegt der verwendete Aggregations-Operator entscheidenden Beschränkungen: Für die Produkt-Norm ist der NRA-Algorithmus beispielsweise gänzlich ungeeignet, da die Gesamt-Relevanz nicht nach unten begrenzt ist. Auch wenn ein Datensatz auf einem Stream bereits mit einer Relevanz von 1.0 eingelesen wurde, so kann seine Gesamt-Relevanz immer noch bei 0.0 liegen. In diesem Fall können keine Datensätze vorab ausgegeben werden, bis sie nicht von beiden Streams eingelesen wurden. In dem Fall ist der Algorithmus nahezu identisch mit dem Algorithmus nach Pfeifer/Fuhr.

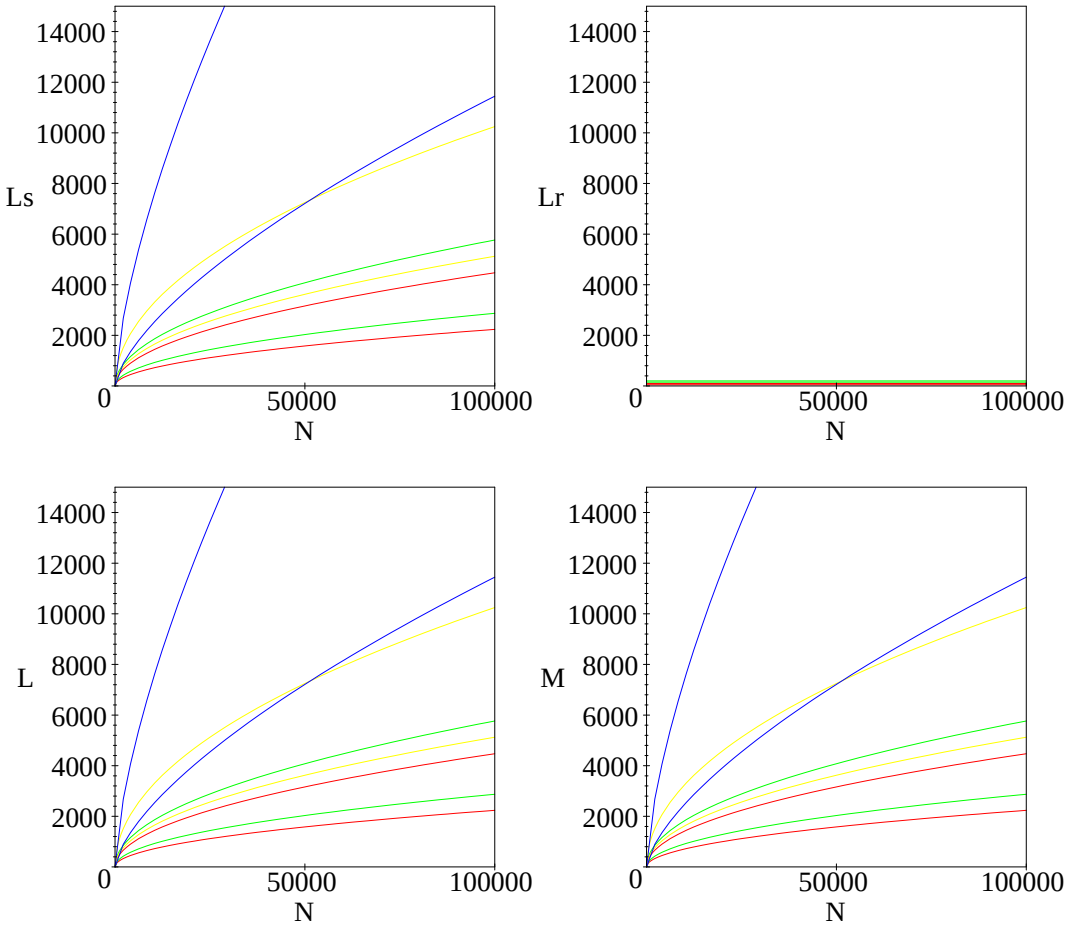


Abbildung 4.21: NRA-Algorithmus: Obere und untere Schranken der Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links) und random accesses $L_r(N,n)$ (oben rechts) sowie der Gesamtlaufzeitkosten $L(N,n)$ (unten links, exemplarisch für $h = 3$) und Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$.

4.3.7 Combined-Algorithmus (CA)

Der in [21] entwickelte Combined-Algorithmus verbindet den Threshold-Algorithmus und den No-Random-Access-Algorithmus zu einem kombinierten Algorithmus, der durch das Wissen über das Verhältnis h der Kosten für einen random access zu einem sorted access die relative Anzahl der random accesses pro sorted access zu optimieren versucht.

Konkret wird der NRA-Algorithmus so modifiziert, dass in jedem h-ten Schritt ein random access durchgeführt wird auf die noch nicht gelesenen Attribute des Objektes mit der höchsten bestmöglichen Relevanz, also dem Objekt, welches am wahrscheinlichsten ausgegeben werden kann. (Für $h = \infty$ wird somit der Spezialfall des NRA abgedeckt.) Genau wie beim NRA-Algorithmus stellen sich die Probleme, dass ein stark erhöhter kalkulatorischer Aufwand nötig ist und der verwendete Aggregationsoperator diversen Beschränkungen unterliegt (z.B. die Produkt-Norm nicht geeignet ist). Ebenso sind wie beim NRA-Algorithmus die konkreten Gesamt-Relevanzen nicht unmittelbar verfügbar, können jedoch durch einen einzigen random access pro Datensatz - also mit minimalen, konstanten Kosten - ermittelt werden. Die Ausweitung des Combined-Algorithmus auf einen echten Streaming-Algorithmus verläuft analog zum NRA. Der formale Algorithmus lautet:

```
INPUT: Stream s1, s2; Integer d;
OUTPUT: Stream out;

List s1Buffer,s2Buffer,outBuffer;
Dataset d1,d2,maxWorstD,maxBestD,d;
Integer counter;
Double r1,r2;

WHILE NOT out.closed() DO

  // naechsten (noch nicht vom anderen Stream
  // gelesenen) Datensatz lesen:

  IF s1.currentRelevance>s2.currentRelevance THEN
    DO
      d1:=s1.read();
    WHILE outBuffer.contains(d1.id);
    r1:=d1.relevance;
    s1Buffer.add(d1);
  ELSE
    DO
      d2:=s2.read();
    WHILE outBuffer.contains(d2.id);
    r2:=d2.relevance;
    s2Buffer.add(d2);
  END IF
```

```
// worst- und best-relevances berechnen (nur jeden d-ten Schritt):

counter:=counter+1;

IF counter MOD d == 0 THEN

  FOR EACH d1 IN s1Buffer DO
    IF s2Buffer.contains(d1.id) THEN
      d2:=s2Buffer.getElement(d1.id);
      d1.worstRelevance:=d1.bestRelevance:=
        agg(d1.relevance,d2.relevance);
    ELSE
      d1.worstRelevance:=agg(d1.relevance,0);
      d1.bestRelevance:=agg(d1.relevance,r2);
    END IF
  END FOR

  FOR EACH d2 IN s2Buffer DO
    IF s1Buffer.contains(d2.id) THEN
      d1:=s1Buffer.getElement(d2.id);
      d2.worstRelevance:=d2.bestRelevance:=
        agg(d1.relevance,d2.relevance);
    ELSE
      d2.worstRelevance:=agg(d2.relevance,0);
      d2.bestRelevance:=agg(d2.relevance,r1);
    END IF
  END FOR

END IF

// Datensatz mit maximaler worst-relevance suchen:

FOR EACH d IN s1Buffer OR IN s2Buffer DO
  IF (d.worstRelevance>maxWorstD.worstRelevance
      OR (d.worstRelevance==maxWorstD.worstRelevance AND
          d.bestRelevance>maxWorstD.bestRelevance) ) THEN
    maxWorstD:=d;
  END IF
END FOR

// Datensatz mit maximaler best-relevance suchen:

FOR EACH d IN s1Buffer OR s2Buffer DO
  IF (d.id!=maxWorstD.id AND
```

```
      d.bestRelevance>maxBestD.bestRelevance) THEN
    maxBestD:=d;
  END IF
END FOR

// random access jeden h-ten Schritt:

step:=step+1;
IF step/2 MOD h = 0 THEN
  IF NOT s1Buffer.contains(maxBestD) THEN
    maxBestD.worstRelevance:=maxBestD.bestRelevance:=
      agg(maxBestD.relevance,s1.randomAccess());
  ELSE IF NOT s2Buffer.contains(maxBestD) THEN
    maxBestD.worstRelevance:=maxBestD.bestRelevance:=
      agg(maxBestD.relevance,s2.randomAccess());
  END IF
END IF

// Datensatz mit hoechster Relevanz wenn moeglich auf
// Outputstream schreiben:

IF maxWorstD.worstRelevance>=maxBestD.bestRelevance THEN
  outBuffer.add(maxWorstD);
  s1Buffer.remove(maxWorstD);
  s2Buffer.remove(maxWorstD);
  out.write(maxWorstD);
END IF

END WHILE
```

Die Kostenmaße $L_s(N,n)$ und $M(N,n)$ haben dieselben Schranken wie die in Kapitel 4.3.6 für den NRA-Algorithmus hergeleiteten Kostenmaße. $L_r(N,n)$ ist definitionsgemäß festgelegt durch $L_r(N,n) = (dim-1) * \frac{1}{h} * L_s(N,n)$. Im worst case liegen die Kosten so hoch wie beim NRA, im Mittel jedoch näher an der unteren Schranke. Für die vier Verteilungen gilt also:

Gleichverteilung:

$$\underline{L_s}(N,n) = \sqrt{2Nn} \qquad \overline{L_s}(N,n) = \sqrt{8Nn}$$
$$\underline{L_r}(N,n) = \frac{1}{h} * \sqrt{2Nn} \qquad \overline{L_r}(N,n) = \frac{1}{h} * \sqrt{8Nn}$$
$$\underline{M}(N,n) = \sqrt{2Nn} - n \qquad \overline{M}(N,n) = \sqrt{8Nn} - n$$

Normalverteilung:

$\underline{L_s}(N,n) = \sqrt{3.3Nn}$ $\quad$ $\overline{L_s}(N,n) = \sqrt{13.3Nn}$

$\underline{L_r}(N,n) = \frac{1}{h} * \sqrt{3.3Nn}$ $\quad$ $\overline{L_r}(N,n) = \frac{1}{h} * \sqrt{13.3Nn}$

$\underline{M}(N,n) = \sqrt{3.3Nn} - n$ $\quad$ $\overline{M}(N,n) = \sqrt{13.3Nn} - n$

Exponentialverteilung:

$\underline{L_s}(N,n) = \sqrt{10.5nN}$ $\quad$ $\overline{L_s}(N,n) = \sqrt{42nN}$

$\underline{L_r}(N,n) = \frac{1}{h} * \sqrt{10.5nN}$ $\quad$ $\overline{L_r}(N,n) = \frac{1}{h} * \sqrt{42nN}$

$\underline{M}(N,n) = \sqrt{10.5nN} - n$ $\quad$ $\overline{M}(N,n) = \sqrt{42nN} - n$

3D-Gleichverteilung:

$\underline{L_s}(N,n) = \sqrt[3]{6N^2n}$ $\quad$ $\overline{L_s}(N,n) = \sqrt[3]{162N^2n}$

$\underline{L_r}(N,n) = \frac{2}{h} * \sqrt[3]{6N^2n}$ $\quad$ $\overline{L_r}(N,n) = \frac{2}{h} * \sqrt[3]{162N^2n}$

$\underline{M}(N,n) = \sqrt[3]{6N^2n} - n$ $\quad$ $\overline{M}(N,n) = \sqrt[3]{162N^2n} - n$

Abbildung 4.22 zeigt die Graphen dieser Kostenfunktionen.

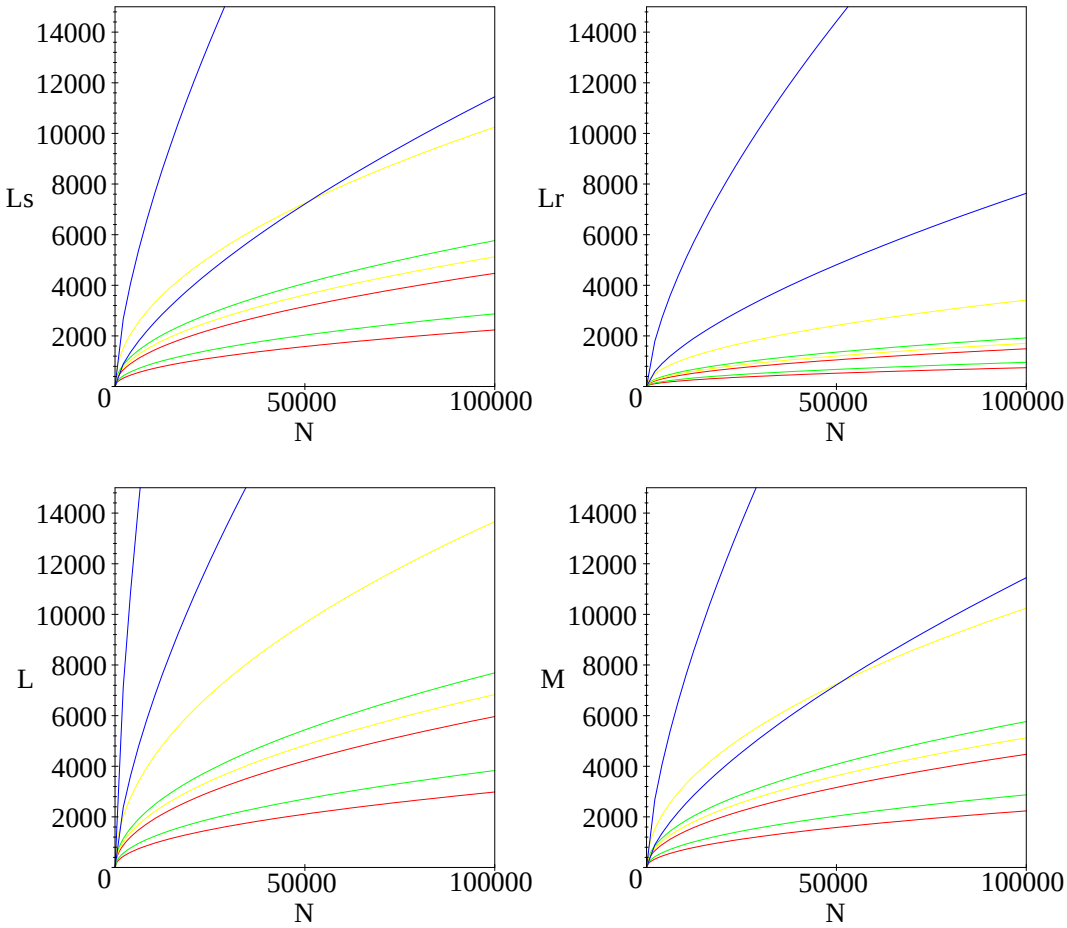


Abbildung 4.22: Combined-Algorithmus: Obere und untere Schranken der Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) sowie Gesamtlaufzeitkosten $L(N,n)$ (unten links) und Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$ und $h = 3$.

4.3.8 Discrete-Streams-Algorithmus (DSA)

In diesem Kapitel wird ein Verfahren entwickelt, welches erfolgreich im Rechercheassistenten [57] eingesetzt wird. Es folgt einem anderen Ansatz als die Verfahren nach Pfeifer/Fuhr oder Fagin und ist eher ein Hybrid aus diesen Algorithmen und den Range-Queries.

Das Ziel dieses Algorithmus ist es, keine unnötigen sorted und random accesses durchzuführen und trotzdem mit minimalem Speicherverbrauch auszukommen. Dadurch wird ein optimales Skalierungsverhalten hinsichtlich der Datenbankgröße erreicht. Erkauft werden diese Vorteile ähnlich wie bei den Range-Queries durch eine höhere Anzahl auszuführender Datenbankanfragen. Die essenziellen Nachteile des Range-Queries-Algorithmus werden dabei jedoch überwunden.

Die grundlegende Idee (bei zwei zu verknüpfenden Attributen) ist es, nicht wie bei Pfeifer/Fuhr oder Fagin für jedes der beiden Attribute einen nach diesem Attribut sortierten Stream von der Datenbank anzufordern, sondern die kontinuierliche Relevanz *eines* Attributes in verschiedene Klassen zu diskretisieren. Für jede dieser Klassen wird eine Anfrage formuliert, deren Ergebnis nach dem *anderen* Attribut sortiert ist. Dabei muss, wie im Folgenden erläutert, nur ein kleiner Teil - im Idealfall eine - dieser diskretisierten Anfragen ausgeführt werden.

Für $k > 2$ zu verknüpfende Attribute wird dieses Prinzip analog ausgeweitet: Es werden $k-1$ der Attribute diskretisiert und die Streams nach dem verbleibenden Attribut sortiert.

Für das bereits häufig verwendete Beispiel der Gebrauchtwagen-Datenbank, in der nach *möglichst neuen und möglichst günstigen* Kraftfahrzeugen gesucht werden soll, würden z.B. die folgenden Anfragen generiert werden (SQL-Syntax):

```
SELECT * FROM gebrauchtwagen
 WHERE baujahr>2000                    ORDER BY preis;

SELECT * FROM gebrauchtwagen
 WHERE baujahr>1998 AND baujahr<=2000 ORDER BY preis;

SELECT * FROM gebrauchtwagen
 WHERE baujahr>1996 AND baujahr<=1998 ORDER BY preis;

SELECT * FROM gebrauchtwagen
 WHERE baujahr>1994 AND baujahr<=1996 ORDER BY preis;

...
```

Jede dieser Anfragen liefert einen Stream, von denen jedoch zunächst nur der erste geöffnet wird.Von diesem werden Datensätze eingelesen, bis zu dem Punkt, ab dem eine bestimmte Gesamt-Relevanz nicht mehr erreicht werden kann. Für alle eingelesenen Datensätze kann die endgültige Relevanz mit Hilfe des verwendeten Fuzzy-Quantors berechnet werden. Zu

berücksichtigen ist, wie auch im folgenden Beispiel gezeigt, dass diese kombinierten *Gesamt*-Relevanzen keineswegs einer absoluten Ordnung innerhalb der einzelnen Streams unterliegen. Vielmehr kann es zu einer partiellen Umsortierung kommen, innerhalb derer einzelne Datensätze vertauscht sind. (Dies liegt an der Varianz der diskretisierten Relevanzen innerhalb einer Quantisierungsstufe, welche durch einen random oder direct access exakt ermittelt werden können.) Da durch das Abbruchkriterium jedoch sichergestellt ist, dass zu diesem Zeitpunkt alle Datensätze bis hin zu der Grenz-Relevanz eingelesen sind, können alle Datensätze mit einer größeren Relevanz sortiert und ausgegeben werden.

Wurde noch nicht die gewünschte Anzahl an Ergebnissen produziert, wird im folgenden Schritt der zweite Stream geöffnet, und es werden parallel von beiden Streams Datensätze gelesen, bis wiederum die mit dem Sortierkriterium (Preis) korrespondierende Relevanz soweit gesunken ist, dass die Gesamt-Relevanz einen neuen, unteren Grenzwert erreicht. Wieder kann die endgültige Relevanz für die eingelesenen Datensätze berechnet werden und alle Datensätze mit einer größeren Relevanz als der aktuellen Grenz-Relevanz ausgegeben werden. Iterativ wird so lange fortgefahren, bis die gewünschte Anzahl an Datensätzen ausgegeben ist.

Abbildung 4.23 zeigt eine exemplarische Verteilung und die geöffneten und zu einem Teil eingelesenen Streams für den Fall von 2 Attributen. Abbildung 4.24 stellt den analogen Fall für 3 Attribute dar. (Jeweils mit dem arithmetischem Mittel als Aggregations-Operator). Das Verfahren wird noch einmal konkret für das zweidimensionale Beispiel verdeutlicht:

Stream 1 (Bj. 2001-2002 $\hat{=}\mu_{Datum} \in]0.8..1.0]$):

a	b	c	
400\$ → $\mu_{Preis} = 0.9$ 2002 → $\mu_{Datum} = 1.0$ $\mu_{agg} = 0.95$	600\$ → $\mu_{Preis} = 0.7$ 2001 → $\mu_{Datum} = 0.9$ $\mu_{agg} = 0.8$	1500\$ → $\mu_{Preis} = 0.2$ 2002 → $\mu_{Datum} = 1.0$ $\mu_{agg} = 0.6$	...

Stream 2 (Bj. 1999-2000 $\hat{=}\mu_{Datum} \in]0.6..0.8]$):

d	e	f	
300\$ → $\mu_{Preis} = 1.0$ 1999 → $\mu_{Datum} = 0.7$ $\mu_{agg} = 0.85$	450\$ → $\mu_{Preis} = 0.85$ 1999 → $\mu_{Datum} = 0.7$ $\mu_{agg} = 0.775$	1000\$ → $\mu_{Preis} = 0.3$ 2000 → $\mu_{Datum} = 0.8$ $\mu_{agg} = 0.55$	...

Stream 3 (Bj. 1996-1998 $\hat{=}\mu_{Datum} \in]0.4..0.6]$):

g	h	i	
300\$ → $\mu_{Preis} = 1.0$ 1997 → $\mu_{Datum} = 0.5$ $\mu_{agg} = 0.75$	350\$ → $\mu_{Preis} = 0.95$ 1998 → $\mu_{Datum} = 0.6$ $\mu_{agg} = 0.775$	600\$ → $\mu_{Preis} = 0.7$ 1998 → $\mu_{Datum} = 0.6$ $\mu_{agg} = 0.65$	...

Wenn zunächst nur der erste Stream geöffnet ist, liegt die Grenz-Relevanz bei $\frac{0.8+1.0}{2} = 0.9$. Ab Datensatz c ist klar, dass diese Relevanz auf Stream 1 nicht mehr erreicht werden kann. Datensatz a, der eine höhere Gesamt-Relevanz hat, kann also ausgegeben werden.

Nach dem Öffnen des zweiten Streams sinkt die Grenz-Relevanz auf $\frac{0.6+1.0}{2} = 0.8$. Auf Stream 1 kann auch diese Relevanz nicht mehr erreicht werden; Daher müssen hier keine

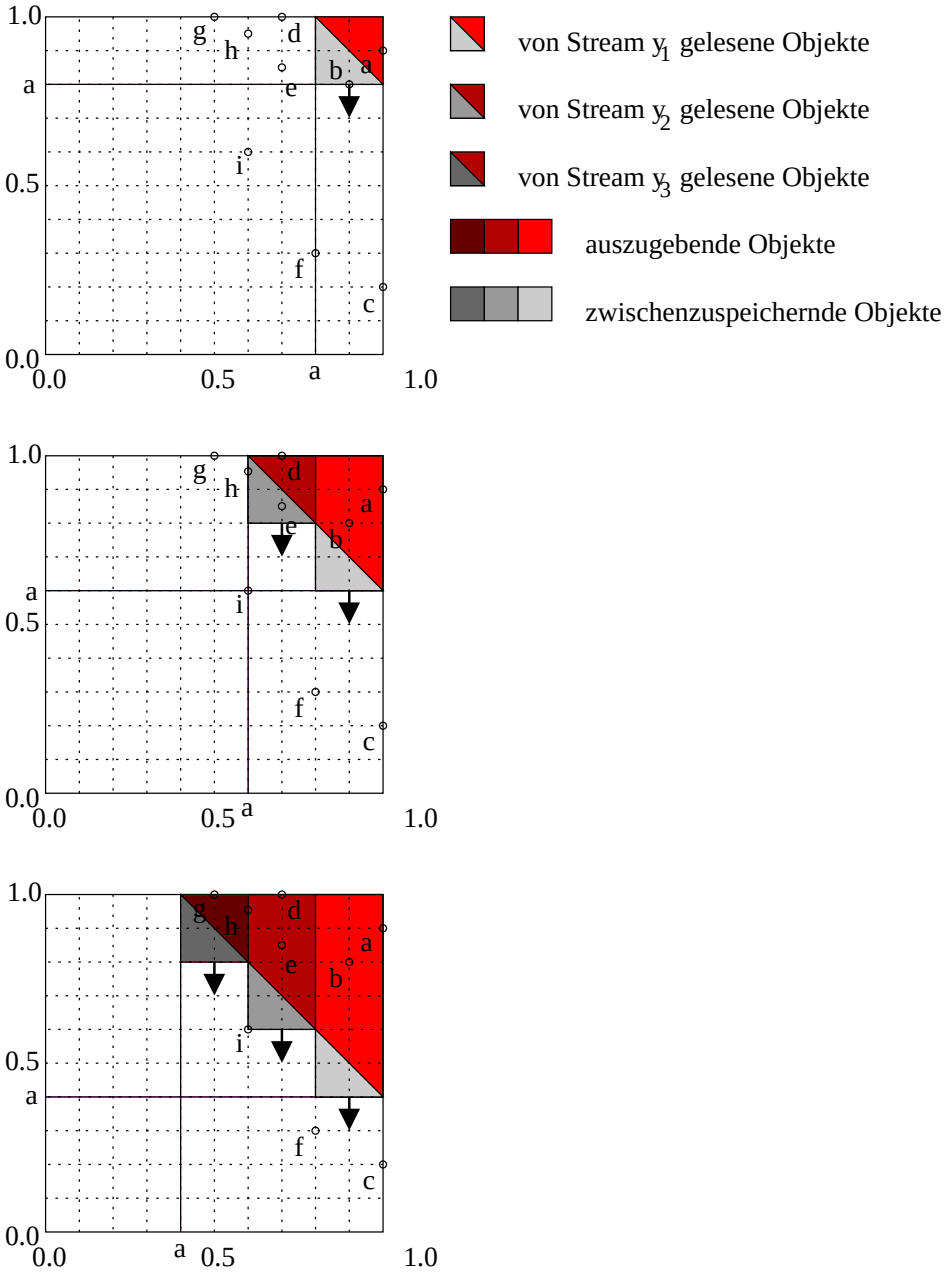


Abbildung 4.23: Exemplarische Verteilung von Datensätzen mit markierten Bereichen der eingelesenen und ausgegebenen Datensätze für den Discrete-Streams-Algorithmus in verschiedenen Stadien. (Quantisierung über x, Sortierung nach y.)

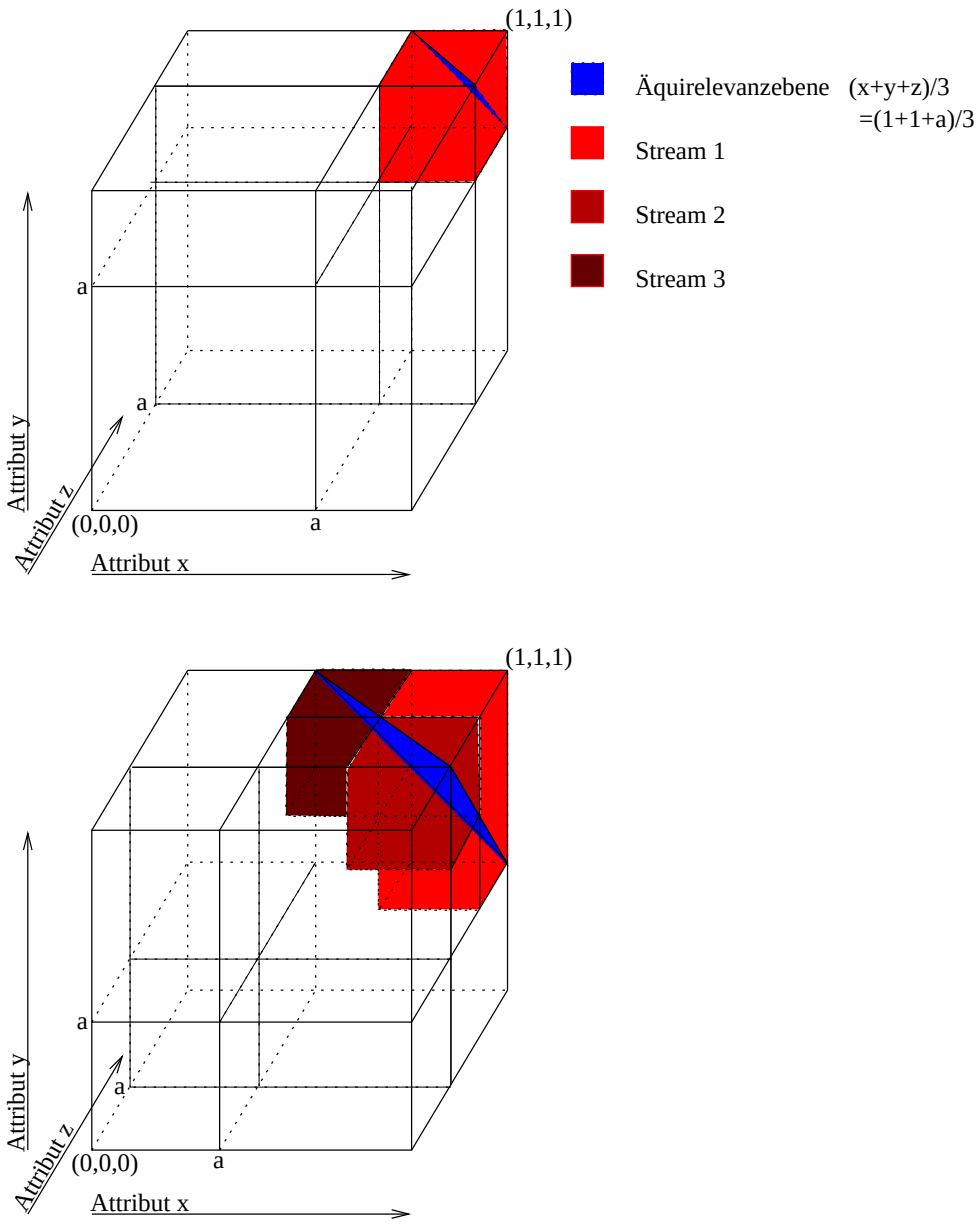


Abbildung 4.24: Dreidimensionale Verteilung von Datensätzen mit markierten Bereichen der eingelesenen und ausgegebenen Datensätze für den Discrete-Streams-Algorithmus in verschiedenen Stadien. (Quantisierung über x und z, Sortierung nach y.)

weiteren Datensätze gelesen werden. Stream 2 muss bis inklusive Datensatz f eingelesen werden. Zu diesem Zeitpunkt können die Datensätze d und b ausgegeben werden.

Mit dem Öffnen des dritten Streams erreicht die Grenz-Relevanz einen Wert von $\frac{0.4+1.0}{2} = 0.7$. Stream 1 und Stream 2 können diesen Wert wieder nicht überschreiten, Stream 3 muss bis inklusive Datensatz i eingelesen werden. Nun können die Datensätze h, e und g ausgegeben werden. (Bei g und h macht sich auch die eingangs erwähnte Tatsache bemerkbar, dass aufgrund der Quantisierung die Sortierung innerhalb der Streams nicht exakt mit der Gesamt-Relevanz korrespondiert.)

Die formale Definition des Algorithmus ist:

```
INPUT: Stream[] streams; // Streams (diskretisiert ueber
        // erstes Attribut, sortiert nach dem zweiten)
       RandomAccess ra; // random access fuer erstes
        // Attribut
       Double[] thresholdRelevances; // untere Grenze der Relevanzen fuer
        // diskretisierte Streams

OUTPUT: Stream out;

SortedList outBuffer;
Integer openStreams;
Dataset dataset;
Double origRelevance;

WHILE NOT out.closed() DO

  // von allen geoeffneten Streams Datensaetze lesen,
  // bis Grenz-Relevanz erreicht:

  FOR i:=1 TO openStreams DO
    DO
      dataset:=streams[i].read();
      origRelevance:=dataset.relevance;
      dataset.relevance:=agg(dataset.relevance,
                             ra.randomAccess(dataset));
      outBuffer.add(dataset);
    UNTIL agg(origRelevance,thresholdRelevances[i-1])<=
          agg(1.0,thresholdRelevances[openStreams]);
  END FOR

  // testen, ob valide Datensaetze vorhanden sind...:

  IF outBuffer.last().relevance
     >= agg(1.0,thresholdRelevances[openStreams]) THEN
```

```
    // ...und diese ausgeben:
    WHILE outBuffer.last().relevance
      >= agg(1.0,thresholdRelevances[openStreams]) DO
        out.push(outBuffer.last());
        outBuffer.removeLast();
    END WHILE

  ELSE

    // ...andernfalls neuen Stream oeffnen:
    openStreams:=openStreams+1;
    streams[openStreams].open();

  END IF

END WHILE
```

Ein noch zu lösendes Problem ist das der Granularität der Quantisierung. Eine zu grobe Granularität führt letztendlich zu denselben Problemen, die sich auch bei den Range-Queries (vgl. Kapitel 4.3.2) stellen, insbesondere dazu, dass sehr viele Datensätze unnötig eingelesen werden. Eine extrem feine Granularität bewirkt jedoch, dass sehr viele Streams geöffnet, also sehr viele Datenbankanfragen ausgeführt werden müssen, was sich ebenfalls negativ auf die Laufzeit auswirkt. Offensichtlich hängt eine sinnvolle Granulierung von mehreren Faktoren, wie der Größenordnung der Datenbank und der Verteilung der Attribute ab. Dies muss im Folgenden bei der Performance-Analyse berücksichtigt werden.

Für einige Anwendungen beantwortet sich die Frage der Granulierung jedoch von selbst: Bei der im bibliographischen Retrieval typischen Anfrage der Form *„möglichst neue Dokumente mit möglichst vielen der Suchbegriffe A,B,C,D"* zerfällt die zweite Teilanfrage ohnehin in 2^s (mit s=Anzahl der Stichworte) bzw. $(a+1)^s$ diskrete Ergebnisklassen (mit a=Anzahl verschiedener Attribute, in denen ein Stichwort vorkommen kann, wie z.B. Titel und Volltext; vgl. Kapitel 4.3.1). In diesem Fall ergibt sich die Granulierung von selbst, da ein Quantisierungsintervall pro Ergebnis-Klasse angenommen wird. Darüber hinaus werden auch keine Datensätze umsonst eingelesen, da aufgrund der diskreten Relevanzen die Sortierung des Datums mit der Sortierung nach aggregierter Gesamt-Relevanz innerhalb jedes Streams eindeutig korrespondiert. Der random access erübrigt sich in diesem Fall auch, da für jeden Stream die untere gleich der oberen Grenze der quantisierten Relevanz ist. Abb. 4.25 veranschaulicht diesen Fall.

Bei Applikationen, bei denen die Granulierung nicht bereits durch die Problemstellung festgelegt ist, ergibt sich die Abschätzung einer geeigneten Anzahl an Quantisierungsstufen wie folgt (die exakte Berechnung folgt später):

Da die Kosten für sorted accesses ($L_s(N,n)$) und random accesses ($L_r(N,n)$) beim Discrete-Streams-Algorithmus minimal sind und üblicherweise der Hauptteil der Gesamtkosten auf

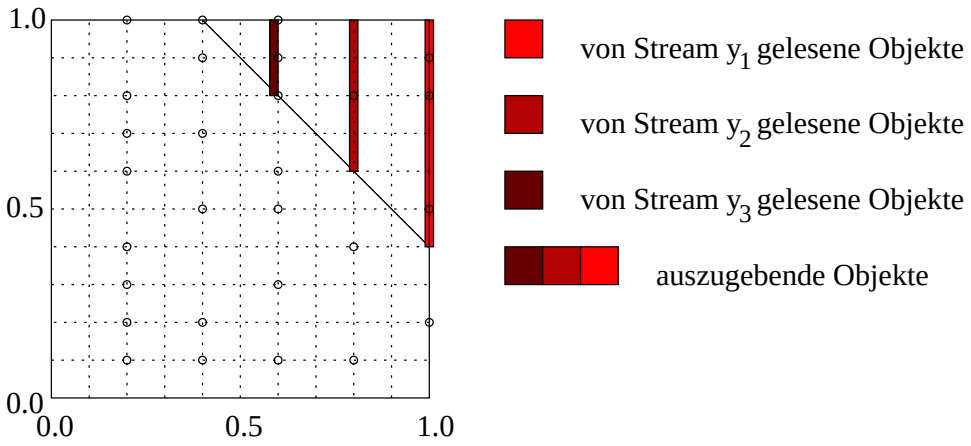


Abbildung 4.25: Diskrete Verteilung von Dokumenten beim bibliographischen Retrieval mit markierten Bereichen der eingelesenen und ausgegebenen Datensätze für den Discrete-Streams-Algorithmus.

das Ausführen von Anfragen zurückzuführen ist, ist es sinnvoll, die Granulierung so abzuschätzen, dass möglichst nur eine Anfrage ausgeführt werden muss, die genau die gewünschte Anzahl an Datensätzen liefert. In diesem Spezialfall ist der Algorithmus ähnlich dem Optimalfall des Range-Queries-Algorithmus, der jedoch aufgrund mangelnder Streaming-Eigenschaften keine weiteren Datensätze liefern kann, bzw. mit der Berechnung von vorne beginnen muss, falls entweder doch nicht genug Datensätze in der Ergebnismenge liegen oder der Benutzer weitere Datensätze anfordert.

Die Anzahl auszuführender Anfragen in Abhängigkeit der Quantisierungsstufen q berechnet sich als $L_q(q,N,n) = ceil((1-a)*q)$ mit $a = A^{-1}(\frac{n}{N})$. Für das optimale q^* muss also gelten:

$$ceil((1 - A^{-1}(\frac{n}{N})) * q^*) = 1$$
$$\Rightarrow \quad q^* = floor(\frac{1}{1 - A^{-1}(\frac{n}{N})})$$

Für die verschiedenen Verteilungen ergibt sich:

Gleichverteilung:
$$q^* = floor(\sqrt{\tfrac{1}{2}\tfrac{N}{n}})$$

Normalverteilung:
$$q^* = floor(\sqrt{0.3\tfrac{N}{n}})$$

Exponentialverteilung:
$q^* = floor(\sqrt{\frac{1}{42}\frac{N}{n}})$

3D-Gleichverteilung:
$q^* = floor(\sqrt[3]{\frac{1}{6}\frac{N}{n}})$

Beispielsweise ergäben sich für Datenbanken mit $N = 100000$ Datensätzen und $n = 10$ angeforderten Ergebnissen je nach Verteilung recht unterschiedliche, optimale Quantisierungen von 71, 55, 16, bzw. 12 Stufen.

Berechnet man die Laufzeitkosten (in Abhängigkeit der Quantisierungsstufen q) exakt, so lässt sich die obige Abschätzung bestätigen, wobei sich außerdem zeigt, dass die Laufzeitkosten um den Bereich der optimalen Quantisierungsstufe q^* eine relativ breite Senke aufweisen, also eine grobe Abschätzung der optimalen Quantisierungsstufe ausreicht:

Der Anteil der eingelesenen Datensätze $G(q,a)$ in Abhängigkeit der Anzahl an Quantisierungsstufen q und der Relevanz a berechnet sich (ähnlich wie das nicht quantisierte $G(a)$ in den vorherigen Kapiteln) hier als Summe über die Integrale der Quantisierungsstufen, wie man sich anhand von Abbildung 4.23 verdeutlichen kann. Man erhält (für zweidimensionale Verteilungen):

$$G(q,a) = \sum_{q'=floor(a*q)}^{q-1} \int_{\frac{q'}{q}}^{\frac{q'+1}{q}} \int_{1-\frac{q'-floor(a*q)+1}{q}}^{1} g(x,y)\,dy\,dx$$

Damit berechnen sich die Laufzeitkosten für sorted access, für random access und die Speicherkosten analog zu den vorherigen Kapiteln:

$$\begin{aligned} L_r(q,N,n) = L_s(q,N,n) &= N * G(q, A^{-1}(\frac{n}{N})) \\ M(N,n) &= L_s(q,N,n) - n = N * G(q, A^{-1}(\frac{n}{N})) - n \end{aligned}$$

Die Gesamtkosten $L(q,N,n)$ berechnen dann zu:

$$\begin{aligned} L(q,N,n) &= c_s * L_s(q,N,n) + c_r * L_r(q,N,n) + c_q * L_q(q,N,n) \\ &= (c_s + c_r) * N * G(q, A^{-1}(\frac{n}{N})) + c_q * ceil((1 - A^{-1}(\frac{n}{N})) * q) \end{aligned}$$

Diese Funktion ist in Abbildung 4.26 in Abhängigkeit von N und q (für eine Gleichverteilung $g(x,y)$) für $n = 1$, $n = 10$ und $n = 100$ dargestellt. Die Teil-Kosten für das Ausführen der

Anfragen steigen sprunghaft mit zunehmender Anzahl an auszuführenden Anfragen an und haben ein negatives Skalierungsverhalten bezüglich N. Das umgekehrte Bild zeigt sich in den Teil-Kosten für das Einlesen der Datensätze. Bei extrem kleiner Anzahl an Quantisierungsstufen steigt dieser Wert schnell an, da viele unnötige Datensätze eingelesen werden müssen (im Extremfall $q = 1$ alle Datensätze). In der dargestellte Summe der Gesamtkosten ist die breite Senke um die optimale Quantisierung q^* besonders deutlich für kleine n zu erkennen. Für größere n ist die Senke entsprechend kleiner.

Der entscheidende Vorteil des Discrete-Streams-Algorithmus ist, dass im Gegensatz zu den reinen Streaming-Algorithmen der vergangenen Kapitel - für die optimale Quantisierung q^* - die Laufzeitkosten nahezu unabhängig von der Größe der Datenbank sind. Dies ist plausibel, wenn man bedenkt, dass der Discrete-Streams-Algorithmus kaum überflüssige Zugriffe durchführt. Im Gegensatz dazu führen die reinen Streaming-Algorithmen sehr viele Zugriffe auf Datensätze durch, die überhaupt nicht ausgegeben werden, wobei die Anzahl dieser Zugriffe mit der Größe der Datenbank wächst.

Der Discrete-Streams-Algorithmus erkauft diesen Vorteil durch die Quantisierung damit, dass mehrere Datenbankanfragen ausgeführt werden müssen, wobei aber die Anzahl auszuführender Anfragen bei einer guten Abschätzung der Verteilung sehr klein ist und insbesondere kaum von der Größe der Datenbank abhängt. (Eine entsprechende Infrastruktur vorausgesetzt, können diese Anfragen auf mehrere Datenbank-Server verteilt bzw. parallelisiert werden, was den Nachteil wieder nivelliert.) Allerdings sollte beim Discrete-Streams-Algorithmus die Verteilung der Attribute in etwa bekannt sein, um die optimale Quantisierung einschätzen zu können. Diese kann jedoch heuristisch durch Stichproben ermittelt werden.

Im Unterschied zu den in den vorigen Kapiteln vorgestellten Algorithmen ist also die neue Kostengröße L_q entscheidend, d.h. ein direkter Vergleich hängt von diesem zusätzlichen Parameter ab. Die bei den anderen Algorithmen entscheidenden Kostengrößen L_s und L_r fallen beim DSA kaum ins Gewicht. Entscheidend für einen direkten Vergleich ist also das Verhältnis der Kosten für einen einzelnen sorted, bzw. random access c_s und c_r zu den Kosten für die Ausführung einer Anfrage c_q. Kapitel 4.3.10 wird u.a. diesen Zusammenhang mit konkreten Zeitmessungen abschließend untersuchen.

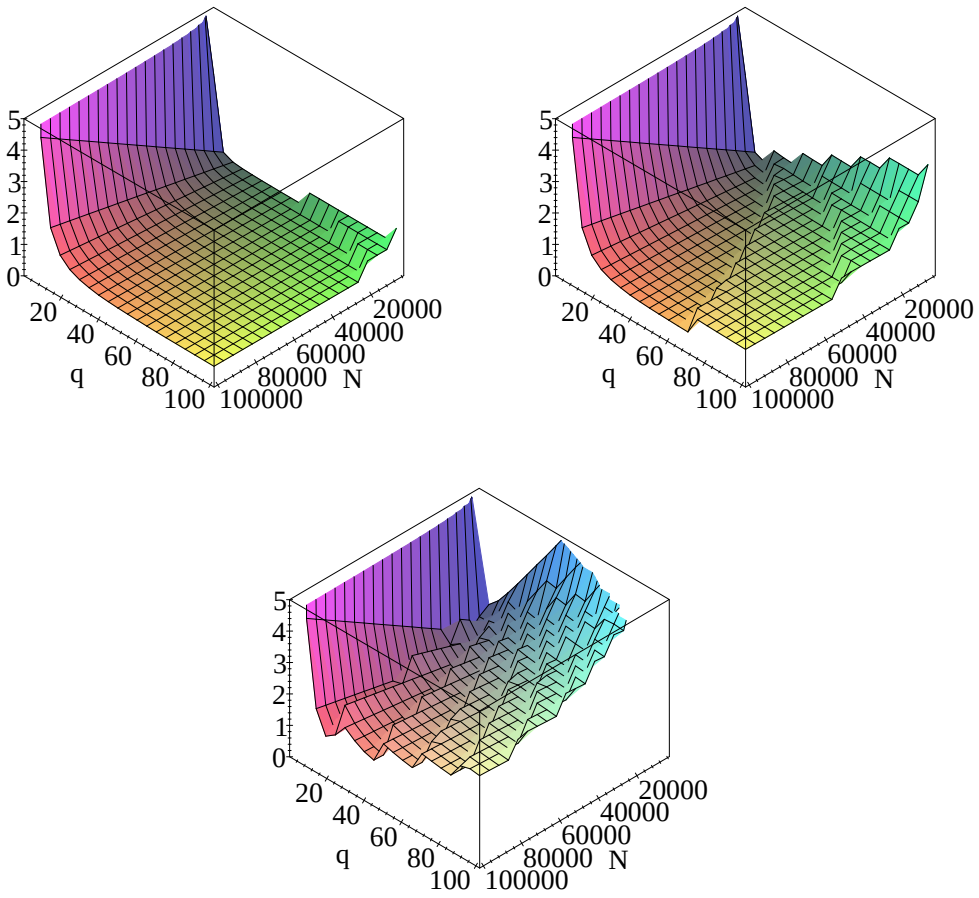


Abbildung 4.26: Gesamtkosten $L(q,N,n)$ für den Discrete-Streams-Algorithmus in Abhängigkeit der Anzahl der Quantisierungsstufen q und der Größe der Datenbank N für gleichverteilte Attribute und $n = 1, 10$ bzw. 100. Die Senke um q^* ist für kleine n extrem breit, für große n wird sie schmaler.

4.3.9 Weitere Varianten von Streaming-Algorithmen

Die bisher vorgestellten Algorithmen von Pfeifer/Fuhr und Fagin, sowie der TA, NRA und CA sind in der Literatur auch unter anderen Namen und in leichten Varianten zu finden (vgl. z.B. [10, 14, 20]).

In [14] werden beispielsweise in einem Vorverarbeitungsschritt die Attribute auf bestimmte Intervalle eingeschränkt, um die Trefferanzahl zu reduzieren. Jedoch führt dieses Verfahren zwangsläufig zu denselben Problemen, wie die Range-Queries (Kapitel 4.3.2).

Andere Varianten wie [10] nehmen eine gewisse Fehlertoleranz in Kauf, die sich in einer nicht unbedingt vollständig korrekten Sortierung des Ergebnisses äußert. Performance wird hier durch Approximation erkauft.

Diese Verfahren weisen jedoch dasselbe Skalierungsverhalten ($O(\sqrt{nN})$ für zwei Attribute bzw. $O(\sqrt[3]{nN^2})$ für drei Attribute) und sogar ein im Wesentlichen identisches, quantitatives Laufzeitverhalten auf (konstante Faktoren nahe 1.0).

Dass durch die reinen Streaming-Ansätze eine untere Schranke nicht unterschritten werden kann, ist dabei plausibel: Die auszugebenden Datensätze müssen zumindest von einem der Streams eingelesen sein. Andernfalls müsste ein Algorithmus „gut raten" bzw. „wild guesses" benötigen, wie auch in [21] dargelegt wird. Abbildung 4.27 visualisiert dieses Problem: Sollten tatsächlich mehr Datensätze ausgegeben werden, so gehörten dazu auch Datensätze, deren Relevanzen ausschließlich durch zufällig gelungene random accesses berechnet werden können. In [10] wird eben dieser in Abbildung 4.27 gelb gekennzeichnete Bereich nicht, bzw. zu spät ausgegeben, wobei diese Datensätze dann genau die eingangs angesprochene Fehlerwahrscheinlichkeit repräsentieren. Dass dadurch - sofern dieser Bereich, und damit die Fehlerwahrscheinlichkeit nicht zu groß werden soll - nur ein marginaler Performance-Gewinn erzielt wird, ist offensichtlich.

Ohne diese Fehlertoleranz berechnet sich die untere Schranke für die sorted accesses reiner Streaming-Algorithmen damit zu:

$$\underline{L_s}(N,n) = N * G(a)$$
$$mit\ a' = A^{-1}(\frac{n}{N})$$
$$mit\ \frac{a'+1}{2} = \frac{a+a}{2} \Leftrightarrow a = \frac{a'+1}{2}\ fuer\ dim = 2$$
$$mit\ \frac{a'+1+1}{3} = \frac{a+a+a}{3} \Leftrightarrow a = \frac{a'+2}{3}\ fuer\ dim = 3$$

Damit ergibt sich konkret:

Gleichverteilung:
$$\underline{L_s}(N,n) = \sqrt{2Nn}$$

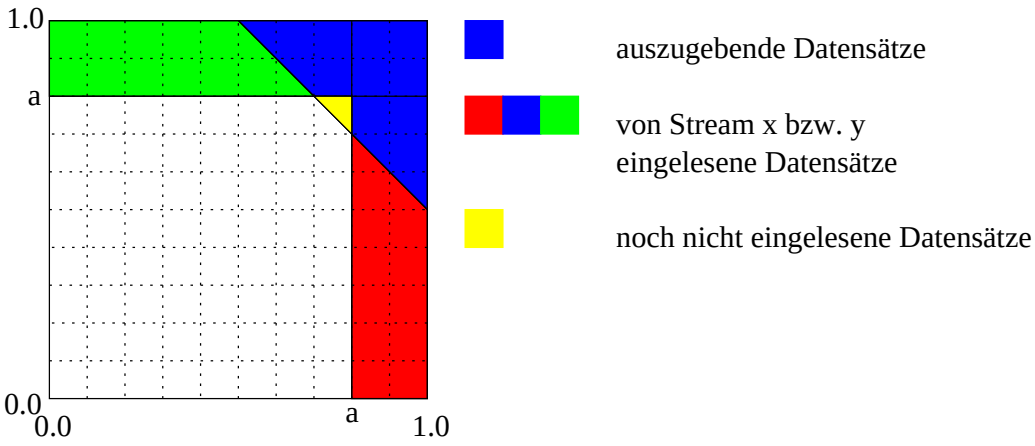


Abbildung 4.27: Fehlerwahrscheinlichkeit bei unterschreiten der unteren Schranke für die sorted accesses. Die Datensätze im gelben Bereich werden nicht bzw. zu spät ausgegeben.

Normalverteilung:
$\underline{L_s}(N,n) = \sqrt{3.3Nn}$

Exponentialverteilung:
$\underline{L_s}(N,n) = \sqrt{10.5nN}$

3D-Gleichverteilung:
$\underline{L_s}(N,n) = \sqrt[3]{6N^2n}$

Diese Graphen sind in Abbildung 4.28 dargestellt. Spielraum für Optimierungen bietet sich darüber hinaus lediglich für den Speicherverbrauch $M(N,n)$ und die random accesses $L_r(N,n)$. So weisen die in den vergangenen Kapiteln (4.3.3-4.3.7) vorgestellten Algorithmen dann auch in diesen beiden Größen die unterschiedlichsten Skalierungsverhalten auf. Aber bereits die Laufzeitkosten für die sorted accesses mit quadratwurzelförmigem Skalierungsverhalten zur Datenbankgröße N (bzw. das Wachstum mit $\sqrt[3]{N^2}$ für drei Attribute) setzen in der Praxis eine Grenze.

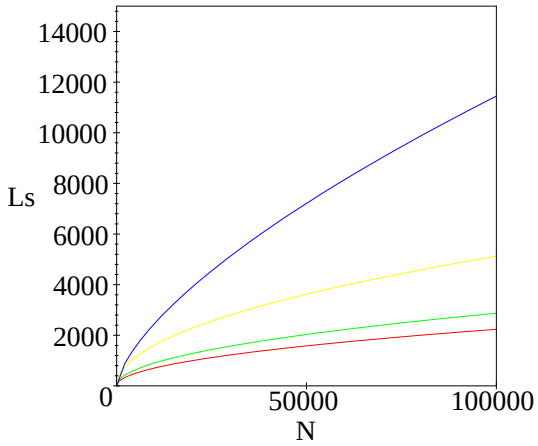


Abbildung 4.28: Die untere Schranke der Laufzeitkosten für sorted accesses $L_s(N,n)$ setzt bereits praktische Grenzen für die reinen Streaming-Algorithmen. $(n = 25)$

4.3.10 Direkter Vergleich der Algorithmen

In den vergangenen Kapiteln wurden verschiedene Verfahren für die Verknüpfung von Fuzzy-Streams vorgestellt. Die reinen Streaming-Verfahren (Kapitel 4.3.4-4.3.7) sind dabei aufgrund ihrer ähnlichen Voraussetzungen und Prämissen sehr gut vergleichbar.

Tabelle 4.2 zeigt eine Übersicht über die Kosten, die die Algorithmen verursachen. Die Anzahl an sorted accesses $L_s(N,n)$ ist bei den meisten Verfahren ähnlich. Insbesondere ist ihr Skalierungsverhalten zur Anzahl gewünschter Ergebnisse n und Stream-Größe N identisch. Die wesentlichen Unterschiede bestehen in der Möglichkeit und dem Umfang, in dem random accesses ausgenutzt werden, sowie im Speicherverbrauch.

Die letztgenannten Kosten schwanken von keinen random accesses ($L_r(N,n) = 0$) und maximalem Speicherverbrauch ($M(N,n) = L_s(N,n) - n$) (dafür aber auch höheren Kosten für die sorted access $L_s(N,n) > \underline{L_s}(N,n)$) beim Pfeifer-Fuhr-Algorithmus bis hin zum anderen Extrem mit konstanten Speicherkosten ($M(N,n) = const$), dafür aber relativ hohen Kosten für die random accesses ($L_r(N,n) = (dim - 1) * L_s(N,n)$) beim TA. Der Fagin-Algorithmus schneidet bei allen Kosten mittelmäßig ab, wird aber vom Threshold-Algorithmus in allen Bereichen unterboten. Der NRA-Algorithmus lässt sich nur sehr grob abschätzen, hat aber maximal die Kosten vom Pfeifer-Fuhr-Algorithmus. Der Combined-Algorithmus, welcher den Threshold-Algorithmus und den NRA-Algorithmus vereint, sollte durch ein optimales Verhältnis von sorted zu random accesses mit den geringsten *Gesamt*-Laufzeitkosten aufwarten, jedoch ist die theoretische, quantitative Abschätzung nicht exakt genug, um das zu bestätigen. Dies wird daher in Kapitel 4.4 im Praxistest geschehen.

Gleichverteilung

	$L_s(N,n)$	$L_r(N,n)$	$M(N,n)$
Pfeifer-Fuhr-Algorithmus	3.162	0	3.137
Fagin-Algorithmus	2.236	2.211	2.186
Threshold-Algorithmus	1.581	1.581	25
No-Random-Access-Algorithmus	1.581-3.162	0	1.556-3.137
Combined-Algorithmus (h=3)	1.581-3.162	527- 1.054	1.556-3.137
untere Schranke von Streaming-Algo.	1.581	-	-
Discrete-Streams-Algorithmus ($q = q^*$)	52	52	27

Normalverteilung

	$L_s(N,n)$	$L_r(N,n)$	$M(N,n)$
Pfeifer-Fuhr-Algorithmus	4.077	0	4.052
Fagin-Algorithmus	2.850	2.825	2.800
Threshold-Algorithmus	2.031	2.031	25
No-Random-Access-Algorithmus	2.031-4.077	0	2.006-4.052
Combined-Algorithmus (h=3)	2.031-4.077	677-1.359	2.006-4.052
untere Schranke von Streaming-Algo.	2.031	-	-
Discrete-Streams-Algorithmus ($q = q^*$)	54	54	29

Exponentialverteilung

	$L_s(N,n)$	$L_r(N,n)$	$M(N,n)$
Pfeifer-Fuhr-Algorithmus	7.246	0	7.221
Fagin-Algorithmus	5.123	5.098	5.073
Threshold-Algorithmus	3.623	3.623	25
No-Random-Access-Algorithmus	3.623-7.246	0	3.598-7.221
Combined-Algorithmus (h=3)	3.623-7.246	1.208-2.415	3.598-7.221
untere Schranke von Streaming-Algo.	3.623	-	-
Discrete-Streams-Algorithmus ($q = q^*$)	66	66	41

3D-Gleichverteilung

	$L_s(N,n)$	$L_r(N,n)$	$M(N,n)$
Pfeifer-Fuhr-Algorithmus	21.634	0	21.609
Fagin-Algorithmus	11.906	23.761	11.831
Threshold-Algorithmus	7.211	14.422	25
No-Random-Access-Algorithmus	7.211-21.634	0	7.186-21.609
Combined-Algorithmus (h=3)	7.211-21.634	4.807-14.423	7.186-21.609
untere Schranke von Streaming-Algo.	7.211	-	-
Discrete-Streams-Algorithmus ($q = q^*$)	231	231	206

Tabelle 4.2: Übersicht über die Laufzeitkosten $L_s(N,n)$ und $L_r(N,n)$ und Speicherkosten $M(N,n)$ der Streaming-Algorithmen für $N = 50.000$ und $n = 25$. Für den NRA und CA lassen sich nur obere und untere Schranken angeben. Die Kosten des DSA sind für das optimale q^* minimal.

Der Discrete-Streams-Algorithmus hingegen kommt mit einer minimalen und darüber hinaus weitgehend von N unabhängigen Anzahl an Zugriffen aus, verursacht jedoch Quantisierungskosten, die bei den anderen Algorithmen nicht gegeben sind, und die sich in einer erhöhten Anzahl an auszuführenden Anfragen äußern. Für einen direkten Vergleich der reinen Streaming-Algorithmen zu dem Discrete-Streams-Algorithmus ist also das Verhältnis der Kosten für einen einzelnen sorted access c_s zu den Kosten für die Ausführung einer komplexen Datenbankanfrage c_q entscheidend.[3] Abbildung 4.29 zeigt den direkten Vergleich, wobei neben dem Discrete-Streams-Algorithmus nur die in Kapitel 4.3.9 hergeleitete, untere Schranke der reinen Streaming-Algorithmen dargestellt ist. Gezeigt werden jeweils die Laufzeitkosten in Abhängigkeit der Größe der Datenbank N, und der Kosten für die Ausführung einer Anfrage c_q für die verschiedenen Verteilungen und für $c_s = 0.0005$.

Offensichtlich schneidet der Discrete-Streams-Algorithmus in den meisten Fällen besser ab, solange sich die Parameter in einem realistischen Rahmen bewegen. Lediglich bei sehr hohen Kosten für die Ausführung einer Anfrage (c_q) und gleichzeitig sehr geringen Kosten für einen sorted bzw. random access (c_s, c_r) liefern die reinen Streaming-Algorithmen bessere Ergebnisse.

[3] Berücksichtigt werden muss dabei, dass auch für die Streaming-Algorithmen Anfragen ausgeführt werden müssen, wenngleich diese eine deutlich geringere Komplexität aufweisen als die diskretisierten Anfragen. c_q steht hier also für die *Mehr*kosten, die die komplexen, quantisierten Anfragen gegenüber einfachen Anfragen benötigen.

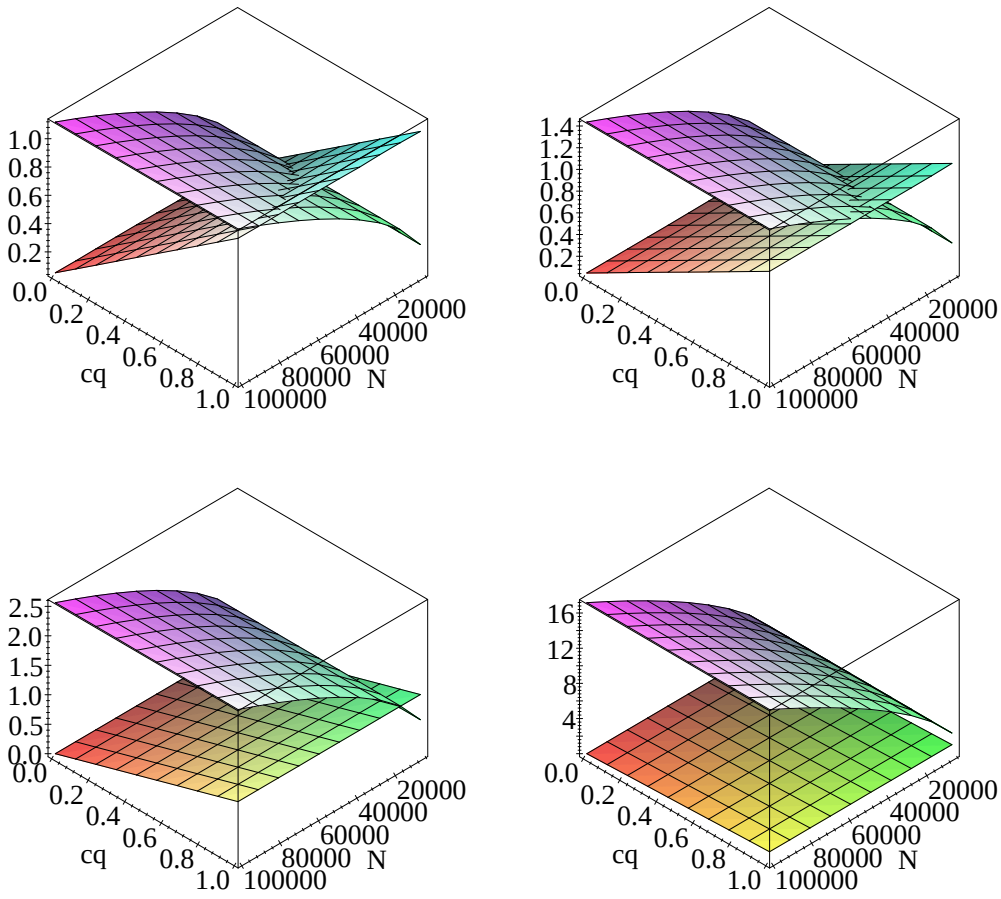


Abbildung 4.29: Laufzeitkosten (in sec) für den Discrete-Streams-Algorithmus (planare Flächen) und untere Schranke der zu erwartenden Laufzeitkosten für die reinen Streaming-Algorithmen (parabelförmige Flächen) in Abhängigkeit der Größe der Datenbank N und der Kosten für die Ausführung einer Anfrage c_q für $c_s = c_r = 0.0005$ und $n = 25$ für Gleichverteilung (oben links), Normalverteilung (oben rechts), Exponentialverteilung (unten links) und 3D-Gleichverteilung (unten rechts). Die Kosten des DSA liegen - trotz einer für diesen Algorithmus eher ungünstigen Schätzung der Parameter - fast immer unter denen der übrigen Algorithmen.

4.4 Praxistests

Zum praktischen Vergleich und zur experimentellen Bestätigung der in den vergangenen Kapiteln angestellten, theoretischen Untersuchung der Algorithmen dienen zwei Szenarien: Zunächst wird die bibliographische Suche, die im Rahmen des Projekts „Rechercheassistent" [57] realisiert wurde, untersucht. Da bei der Stichwortsuche üblicherweise von booleschen bzw. diskreten Zugehörigkeitsgraden ausgegangen werden kann, und nur wenige echte Fuzzy-Attribute (wie z.B. „Erscheinungsdatum möglichst neu") in die Suche einfließen, bietet diese Anwendung nur eingeschränkte Testmöglichkeiten. Daher wird eine zweite Anwendung, eine SQL-Datenbank, die über Datensätze mit statistisch unterschiedlich verteilten Attributen verfügt, untersucht.

Abschließend werden in Kapitel 4.4.3 die Resultate einer Benutzerbefragung dargestellt, die über das Projekt „Rechercheassistent" durchgeführt wurde - dem primären Anwendungsszenario, in dem die Ergebnisse dieser Arbeit unter echten Praxisbedingungen zum Einsatz kommen. An dieser Stelle wird vor allem auch der Zufriedenheitsgrad der Benutzer mit der verwendeten, in Kapitel 3 vorgestellten Suchsemantik deutlich.

4.4.1 Rechercheassistent

Wie bereits in den vergangenen Kapiteln dargestellt, sind beim bibliographischen Retrieval Anfragen typisch, die aus einer Aggregation von booleschen bzw. diskreten Attributen bestehen und darüber hinaus z.B. über ein weiteres Fuzzy-Attribut „Datum möglichst neu" verfügen. Da die Zugehörigkeitsgrade von Dokumenten zu den einzelnen, gesuchten Stichworten üblicherweise boolesch, oder zumindest diskret verteilt sind, sind die Streaming-Verfahren nach Pfeifer/Fuhr etc. zur Aggregation der Stichwort-Mengen nicht sinnvoll einzusetzen. Vielmehr ist die in Kapitel 4.3.1 dargestellte, datenbankseitige Zerlegung in disjunkte Teilergebnisklassen inhärent gegeben:

```
1.00: <keyword1 IN title> AND <keyword2 IN title> AND <keyword3 IN title>
0.90: <keyword1 IN title> AND <keyword2 IN title> AND <keyword3 IN abstract>
      AND NOT <keyword3 in title>
0.80: <keyword1 IN title> AND <keyword2 IN abstract>
      AND NOT <keyword2 in title> AND <keyword3 in title>
...
```

Für die Verknüpfung mit dem weiteren Fuzzy-Attribut „Datum möglichst neu" bietet sich damit unmittelbar der Discrete-Streams-Algorithmus an. Ein direkter Vergleich mit den reinen Streaming-Algorithmen, für die diese Quantisierung einen unnötigen Overhead bedeutet, wäre daher nicht angebracht, da diese zwangsläufig schlechter abschneiden müssen, was informelle Tests in eindeutiger Weise bestätigt haben.

Interessant ist daher vor allem die Performanz bei einer Anfrage mit ausschließlich booleschen Zugehörigkeitsgraden und Fuzzy-Aggregationsoperator (also die reine Fuzzy-Stichwortsuche), ausgeführt über den speziellen, in Kapitel 4.3.1 vorgestellten, konventionellen

Algorithmus im Vergleich zu einer analogen Anfrage mit dem zusätzlichen Attribut „Datum möglichst neu", ausgeführt durch den Discrete-Streams-Algorithmus aus Kapitel 4.3.8. Weiterhin von Interesse ist das Skalierungsverhalten bezüglich der Größe der Datenbank.

Zu diesem Zweck wurden 961 Anfragen, die von Benutzern an den Rechercheassistenten gestellt wurden - aufgeteilt in Klassen nach der Anzahl der eingegebenen Stichworte - auf den Datenbanken „OPAC" mit $N = 1.2\ Mio$ Dokumenten und „JADE IBZ" mit knapp doppelt so vielen Dokumenten ($N = 2.2\ Mio$) ausgeführt, und zwar jeweils einmal alleine als Aggregation der Stichworte und einmal als Aggregation mit dem zusätzlichen Filter-Operator „Datum möglichst neu".

Wie in Kapitel 4.3.8 dargestellt, verursacht der Discrete-Streams-Algorithmus keinen Overhead an sorted oder random accesses; dies konnte in der Praxis bestätigt werden. Der entscheidende Faktor, die durchschnittlich benötigte Anzahl an Datenbankanfragen in Abhängigkeit der verschiedenen Parameter ist in Abbildung 4.30 zu sehen. Demnach steigt die Anzahl an auszuführenden Anfragen erwartungsgemäß mit steigender Anzahl an eingegebenen Stichworten schnell an. Dies ist plausibel, da das Maximum k_{max} an Anfragen bei s Stichworten $2^s - 1$ beträgt und von diesem immer etwa die Hälfte ausgeführt werden muss. (Vgl. auch Kapitel 4.3.1.)

Das eigentlich Interessante sind zwei Aspekte: Zum einen verursacht der zusätzliche Filter-Operator (rechte Diagramme im Vergleich zu linken Diagrammen) kaum einen Mehraufwand. Im Vergleich zu den anderen Streaming-Algorithmen, die stark mit der Anzahl zu verknüpfender Attribute skalieren, ist dies bemerkenswert.

Der andere Aspekt betrifft das Skalierungsverhalten bezüglich der Größe der Datenbank N (Vergleich rote zu grünen Graphen): Sowohl bei OPAC mit 1.2 Mio. Dokumenten als auch bei JADE IBZ mit fast doppelt so vielen (2.2 Mio.) Dokumenten verlaufen die Graphen praktisch identisch. Das Laufzeitverhalten des Discrete-Streams-Algorithmus ist also nahezu unabhängig von der Größe der Datenbank - auch dies ist ein entscheidender Unterschied zu den anderen Streaming-Algorithmen.

In der Praxis muss dabei selbstverständlich berücksichtigt werden, wie stark die zugrundeliegenden booleschen Datenbanken mit dem zu bewältigenden Datenvolumen skalieren, wie lange also die Datenbank für die Ausführung einer Anfrage benötigt. Dieser Aspekt hängt natürlich stark von dem verwendeten DBMS ab, ist jedoch durch eine sinnvolle Implementierung innerhalb eines gemeinsamen Kontextes leicht zu minimieren (was bei der Interaktion des Rechercheassistenten mit dem zugrundeliegenden DBMS auch realisiert wurde). Betrachtet man, welche Teile beim Ausführen einer komplexen Datenbankanfrage die meiste Zeit kosten, so ergibt sich für das verwendete „BRS/Search" folgendes Bild:

- Verbindungsaufbau/Login: 100 ms
- Suche nach einem Stichwort (ohne Trunkierung): ca. 50-500 ms (abhängig von Datenbank, Indexierung und Stichwort)
- Suche nach einem Stichwort (mit Trunkierung): ca. 500-5000 ms (abhängig von Datenbank, Indexierung und Stichwort)

- Verknüpfung bereits berechneter Teilergebnismengen mittels AND, OR und NOT: 100-500 ms (abhängig von Operator, Datenbank, Indexierung und Größe der Ergebnismengen)
- Ausgabe eines Dokumentes: 4-12 ms (abhängig von Datenbank und Größe des Datensatzes)

Werden beispielsweise die beiden Anfragen „A AND B" und „A AND NOT B" unabhängig voneinander ausgeführt, so benötigen die Anfragen jeweils bis zu $100 + 2 * 500 + 500 = 1600\ ms$, zusammen also $3200\ ms$. Innerhalb eines gemeinsamen (BRS-spezifischen) Kontextes, in dem die beiden Anfragen auf die Teilergebnisse der atomaren Anfragen nach den Suchbegriffen „A" und „B" gemeinsam zugreifen können und lediglich die Konjunktion bzw. Negation spezifisch für die jeweilige Anfrage ist, kann eine enorme Zeitersparnis erfolgen. In diesem Fall werden zusammen maximal lediglich $100+2*500+2*500 = 2100\ ms$ benötigt. Im Durchschnitt liegen diese Werte deutlich niedriger.

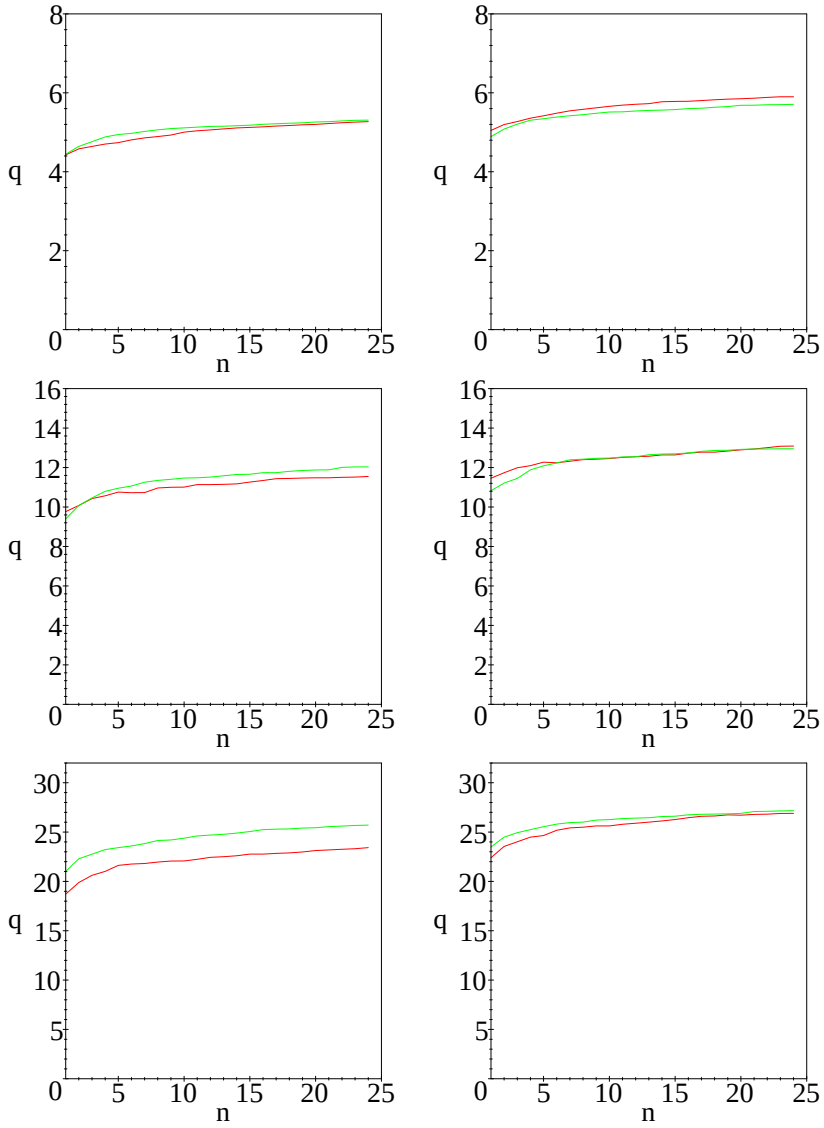


Abbildung 4.30: Mittlere Anzahl auszuführender Datenbankanfragen für $s = 3$, 4 bzw. 5 Stichworte (oben, Mitte, unten) jeweils für reine Aggregation (links) bzw. Aggregation mit zusätzlichem Filter-Attribut (rechts) für die Datenbanken „OPAC" mit $N = 1.2\ Mio$ Dokumenten (rot) und „JADE IBZ" mit $N = 2.2\ Mio$ Dokumenten (grün) in Abhängigkeit der Anzahl an angeforderten Dokumenten n. Der Discrete-Streams-Algorithmus verursacht bei Verwendung eines zusätzlichen Filter-Operators einen nur sehr geringen Overhead und ist darüber hinaus unabhängig von der Größe der Datenbank.

4.4.2 Exemplarische SQL-Datenbank

Die in den vorherigen Kapiteln vorgestellten Algorithmen wurden auf eine SQL-Datenbank angewendet, die Attribute mit allen der untersuchten Verteilungen (zweidimensionale Gleich-, Normal- und Exponentialverteilung sowie dreidimensionale Gleichverteilung) beinhaltete. Es wurden Testläufe mit 10 unterschiedlichen Datenbankgrößen von $N = 10.000$ bis 100.000 Datensätzen durchgeführt und die vorgestellten Kostenmaße gemessen:

- **sorted accesses** $L_s(N, n)$: sequentielle Zugriffe auf die zu verknüpfenden Streams
- **random accesses** $L_r(N, n)$: wahlfreie, Index-/ID-basierte Zugriffe auf die Datensätze in der Datenbank
- **Gesamtlaufzeitkosten** $L(N, n)$: Gewichtete Summe von sorted accesses und random accesses: $L(N, n) = L_s(N, n) + h * L_r(N, n)$. Dabei ist h das Verhältnis der Kosten für einen random zu einem sorted access.
- **Speicherkosten** $M(N, n)$: Anzahl im Speicher zu haltender Datensätze.

Einen entscheidenden Einfluss auf die Gesamtlaufzeitkosten hat dabei der Faktor h, der sehr spezifisch von der Datenbank und Art der Metasuche abhängig ist, und meistens im Bereich von Faktor $3-4$ liegt. In vielen Fällen können die random accesses jedoch durch kostenfreie direct accesses ersetzt werden (vgl. Kapitel 2), wodurch h de facto 0 beträgt. Dies ist immer dann möglich, wenn die zu verknüpfenden Streams von der gleichen Datenquelle stammen, die sämtliche Attribute der Datensätze auf einmal liefern kann.

Aus diesem Grund werden die Kosten zunächst detailiert aufgeschlüsselt werden, und erst zusammenfassend konkrete Zeitangaben erfolgen. Die Gesamtlaufzeitkosten sind demnach exemplarisch zu verstehen.

Die in diesem Test-Szenario gemessenen Kosten bewegen sich ziemlich genau im erwarteten Rahmen, wie in Abbildungen 4.31-4.36 zu sehen ist (vgl. die theoretisch hergeleiteten Ergebnisse in Abb. 4.14-4.22): Der für zwei zu verknüpfende Attribute quadratwurzelförmige Zusammenhang zur Datenbankgröße N ist gut zu erkennen, und auch die konstanten Vorfaktoren stimmen mit den theoretisch hergeleiteten Werten überein.

Interessant sind vor allem die Ergebnisse des NRA-Algorithmus und des Combined-Algorithmus (bei denen nur die Rangfolge der Ergebnisse berechnet wurde, nicht die genaue Relevanz), für die keine exakten, theoretische Ergebnisse vorhersagbar waren, sondern nur obere und untere Schranken angegeben werden konnten.

Der NRA-Algorithmus liegt in allen Kostenmaßen L_s, L_r, L und M nahe der hergeleiteten, oberen Schranke der Abschätzung. Der hohe kalkulatorische Aufwand dieses Algorithmus lohnt sich gegenüber dem ähnlichen Algorithmus von Pfeifer offensichtlich nicht. Auf die Gründe dafür wird später eingegangen.

Die Kosten für die sorted accesses L_s des Combined-Algorithmus liegen erwartungsgemäß unter denen des NRA-Algorithmus. Die zusätzlichen random accesses des - von diesen

abgesehen zum NRA-Algorithmus identischen - Combined-Algorithmus führen tatsächlich dazu, dass früher Datensätze ausgegeben werden können. Für die zweidimensionalen Verteilungen ist dieser Gewinn nur marginal, die sorted accesses L_s können nur geringfügig reduziert werden. Lediglich für die dreidimensionale Verteilung, also bei mehreren zu verknüpfenden Attributen, ist der Vorteil des Combined-Algorithmus bemerkenswert: Während sich der NRA-Algorithmus nahe an der theoretisch hergeleiteten, oberen Grenze befindet, liegt der Combined-Algorithmus im Mittelfeld der Abschätzung.

Zu den sorted accesses L_s kommen beim Combined-Algorithmus jedoch noch die random accesses L_r, die diesen Vorteil - unter den einleitend genannten Bedingungen - u.U. wieder ausgleichen: Die summierten Gesamtkosten L liegen (für $h = 3$) beim Combined-Algorithmus sowohl bei den zwei-, wie auch bei der dreidimensionalen Verteilung entgegen der Erwartung deutlich *über* denen des NRA und sind nahezu identisch mit denen des weit weniger komplexen Threshold-Algorithmus.

Lediglich bei den Speicherkosten ist der Combined-Algorithmus besser als der NRA (wobei dieser Effekt auch nur bei der dreidimensionalen Verteilung spürbar ins Gewicht fällt). Im Vergleich dazu kommt jedoch der Threshold-Algorithmus mit minimalen Speicherkosten aus.

Die Ergebnisse des Discrete-Streams-Algorithmus letztendlich sind wenig überraschend: Definitionsgemäß ist die Anzahl an sorted und random accesses sowie der Speicherverbrauch im Vergleich zu den anderen Algorithmen minimal und darüber hinaus nahezu unabhängig von der Größe der Datenbank N. Der entscheidende Laufzeitfaktor, durch den dieser Vorteil erkauft wird, ist wie bereits erwähnt die wegen der Quantisierung der Attribute erhöhte Anzahl bzw. Komplexität an auszuführenden Datenbankanfragen. Da die Abschätzung der optimalen Quantisierungsstufen bei den bekannten Verteilungen der Attribute relativ genau geschehen konnte, wurde die optimale Quantisierung in vielen Fällen erreicht, so dass eine oder maximal zwei quantisierte Datenbank-Anfragen genügten.

Die Ergebnisse des NRA und des CA, die bisher nur abgeschätzt wurden, werden plausibel, wenn man die Arbeitsweise dieser Algorithmen noch einmal näher betrachtet und ihr Laufzeitverhalten mit dem des Pfeifer-Fuhr-Algorithmus und des Threshold-Algorithmus vergleicht, deren theoretische Betrachtungen auch als obere bzw. untere Schranke dienten:

Die Grundidee des NRA-Algorithmus und des Combined-Algorithmus war, die Relevanz jedes Datensatzes nach oben und unten abzuschätzen, und einen Datensatz auszugeben, wenn klar ist, dass es keinen Datensatz geben kann, der eine höhere Relevanz bekommen kann. Für diese Abschätzung muss eine Fallunterscheidung getroffen werden:

1. Der Datensatz wurde bereits von beiden Streams eingelesen: Die endgültige Relevanz kann dann exakt berechnet werden, die „worst relevance" ist gleich der „best relevance": $\mu = \mu_{worst} = \mu_{best} = Agg(\mu_1, \mu_2)$
2. Der Datensatz wurde erst von einem der beiden Streams eingelesen. Die „worst relevance" berechnet sich dann zu $\mu_{worst} = Agg(\mu_1, 0.0)$

Der erste Fall ist nun exakt der Fall, nach dem auch der Pfeifer-Fuhr-Algorithmus arbeitet (der ja explizit auf diese matches wartet), und braucht daher nicht weiter betrachtet zu werden.

Der zweite Fall ist der, der einen Vorteil gegenüber dem Pfeifer-Fuhr-Algorithmus verspricht. Dieser Vorteil ist allerdings gering, denn die untere Abschätzung ist relativ schlecht: Beispielsweise für das arithmetische Produkt oder die Minimums-Norm als Aggregations-Operator erhält man immer $\mu_{worst} = 0.0$, weswegen im Vergleich zum Pfeifer-Fuhr-Algorithmus überhaupt kein Vorteil entsteht. Aber auch für das oft verwendete und auch in dieser Arbeit benutzte arithmetische Mittel ist die Abschätzung mit $\mu_{worst} = \frac{\mu_1}{2}$ so schlecht, dass erst bei etwa bis zur Hälfte eingelesenen Streams mit signifikanten Vorteilen zu rechnen ist.

Für mehr als zwei zu verknüpfende Attribute tritt der zweite Fall noch weiter in den Hintergrund, da die „worst relevance" $\mu_{worst} = Agg(\mu_1, 0.0, 0.0, ..., 0.0)$ noch wesentlich geringer liegt.

Für den NRA-Algorithmus bedeutet dies, dass er in der Praxis nicht besser arbeitet als der Algorithmus nach Pfeifer/Fuhr. Der erhöhte kalkulatorische Aufwand ist hier also ohne Nutzen.

Der Combined-Algorithmus hingegen benötigte weniger sorted accesses als der NRA und der Pfeifer-Fuhr-Algorithmus. Aufgrund der Tatsache, dass er random accesses ausnutzt, ist seine Arbeitsweise eher mit dem Threshold-Algorithmus, als mit dem Pfeifer-Fuhr-Algorithmus vergleichbar. Im Vergleich zu diesem benötigt er mehr sorted accesses, dafür aber weniger random accesses. Betrachtet man die Ursache dafür genauer, so wird auch dieser Effekt plausibel: Zu einem Zeitpunkt, zu dem eine bestimmte Anzahl an Datensätzen über sorted accesses eingelesen wurde, muss der Threshold-Algorithmus zu jedem eingelesenen Datensatz einen random access auf jedes der anderen Attribute durchführen. Der Combined-Algorithmus hingegen führt nur einen Bruchteil dieser random accesses durch, nämlich nur für die Datensätze, die am wahrscheinlichsten ausgegeben werden können. L_r ist zunächst also sehr viel niedriger. Da jedoch nur für diese relativ wenigen Datensätze random accesses stattfinden, werden einige der Datensätze, die vom Threshold-Algorithmus zu diesem Zeitpunkt bereits ausgegeben werden können, nicht erreicht. Der Combined-Algorithmus muss also einige wenige Iterationsschritte weiterlaufen, wodurch sich die geringfügig höhere Anzahl an sorted accesses erklären lässt.

Der hohe kalkulatorische Aufwand, der bereits beim NRA-Algorithmus nötig war und dort nicht den erhofften Effekt brachte, eine sinnvolle Abschätzung der Relevanzen zu ermöglichen, wirkt sich also auch beim Combined-Algorithmus nicht in der eigentlich intendierten Weise aus, dient aber offensichtlich dazu, eine sinnvolle Entscheidung treffen zu können, welche der random accesses lohnenswert sind, um den Algorithmus vorzeitig terminieren zu lassen. Die Gesamtlaufzeitkosten können dadurch jedoch trotzdem nicht spürbar reduziert werden, da zwar weniger Schleifendurchläufe benötigt werden, in diesen jedoch eben die zusätzlichen random accesses durchgeführt werden.

Die Tests zur Zeitmessung wurden durchgeführt auf einer 100.000 Datensätze umfassenden MySQL-Datenbank (Version 3.23.39) mit Datensätzen der Größe 28 bytes, die auf einem AMD-K7 Athlon mit 600 MHz und 128 MB RAM unter Linux (SuSe 7.2, Distributions-

Standard-Kernel Version 2.4.4) lief. Der Client, auf dem die Aggregation durchgeführt wurde, war vom selben Typ und war über ein 100 MBit-Ethernet mit dem Datenbankserver verbunden. Als Datenbanktreiber wurde der offizielle MySql-JDBC-Treiber, Version 1.3, verwendet. Die Algorithmen liefen unter dem JDK1.2.2 (build 1.2.2_006, green threads, nojit) der Firma „sun". Aufgeschlüsselt nach Aktionen ergeben sich folgende Zeiten:

- Initialisierung eines Streams (Ausführen einer einfachen SELECT-Anfrage[4] ohne Auslesen der Ergebnisse): 835 ms
- Initialisierung eines quantisierten Streams (Ausführen einer SELECT-Anfrage mit WHERE-Clause[5] ohne Auslesen der Ergebnisse): 299 ms
- Initialisierung eines Prepared-Statements[6] für random accesses: 76 ms
- Kosten für einen sorted access (Auslesen eines Datensatzes): 0.253 ms
- Kosten für einen random access (Auslesen eines Datensatzes): 0.932 ms

Über die konkreten Zeiten, die unter der angegebenen Konfiguration für die verschiedenen Algorithmen gemessen wurden, geben die Tabellen 4.3 und 4.4 Aufschluss, für eine Datenbank der Größe $N = 100.000$ mit zwei gleichverteilten Attributen und $n = 25$ angeforderten Datensätzen. Die Tabellen unterscheiden darin, ob random accesses oder direct accesses verwendet werden können.

Die Ergebnisse, die sich bereits aus den Zugriffs-Häufigkeiten (vgl. Abb. 4.31-4.36) abzeichneten, werden durch die konkreten (wenngleich wie geschildert exemplarisch zu verstehenden) Zeitangaben bestätigt. Der Discrete-Streams-Algorithmus hat nur minimale Kosten zu verzeichnen, da er aufgrund des Wissens über die statistische Verteilung der Datensätze eine optimale Quantisierung verwenden konnte. Unter den übrigen, reinen Streaming-Algorithmen ist der Pfeifer-Fuhr-Algorithmus am effizientesten, falls keine direct accesses möglich sind. Die eingesparten sorted accesses bei den anderen Algorithmen zahlen sich in dem Fall aufgrund der hohen Kosten der random accesses nicht aus.

Anders bei den günstigen direct accesses: Hier ist der Threshold-Algorithmus der effizienteste, weil sich die eingesparten sorted accesses in Relation stärker auswirken. NRA und CA sparen wie erwartet kaum Datenbankzugriffe ein, benötigen aufgrund ihres kalkulatorischen Aufwands jedoch zusätzliche Zeit. Der CA schneidet im Vergleich zum NRA auch deswegen deutlich schlechter ab, weil er die Abschätzung der Relevanzen in jedem h-ten Schritt (im Gegensatz zu jedem d-ten Schritt beim NRA) berechnen muss, wodurch der kalkulatorische Aufwand noch weiter steigt.

[4] SELECT * FROM table ORDER BY attribute
[5] SELECT * FROM table ORDER BY attribute1 WHERE attribute2>min AND attribute2<=max
[6] SELECT attribute1 FROM table WHERE id=%variable

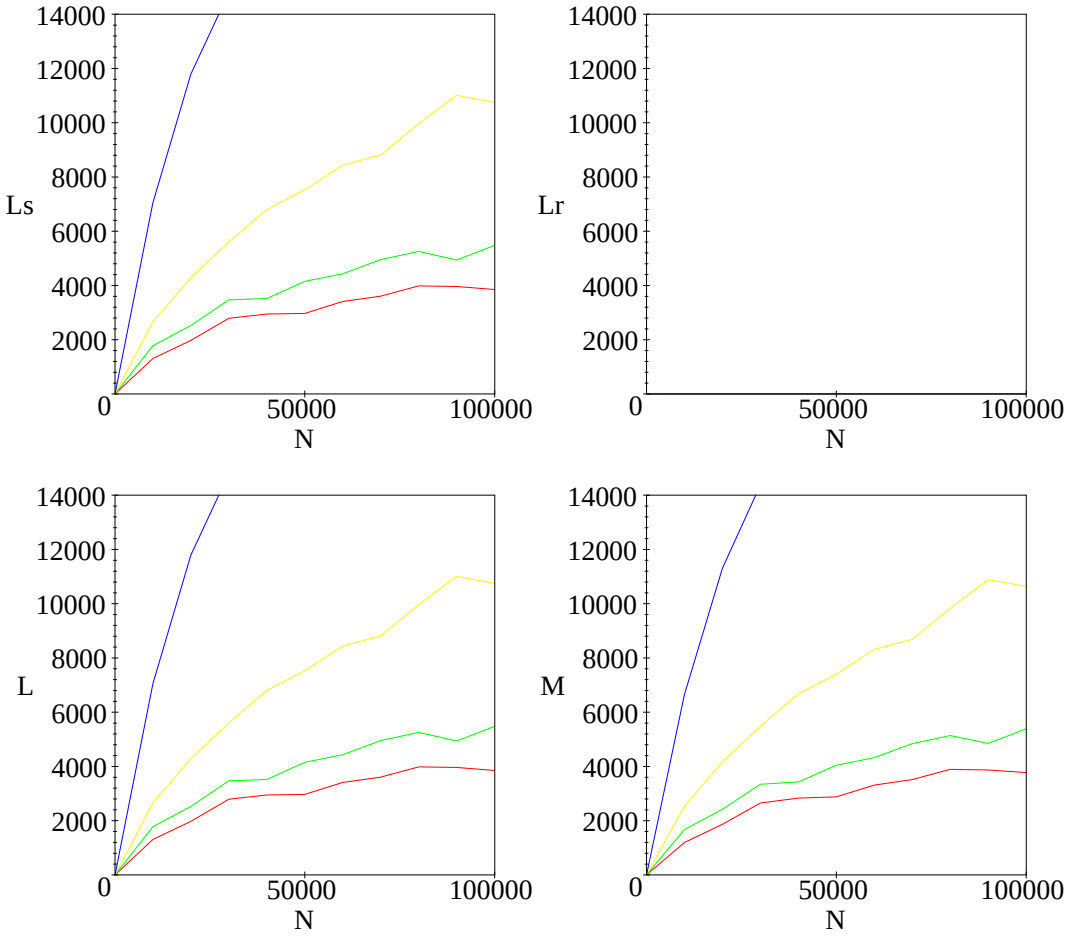


Abbildung 4.31: Pfeifer-Fuhr-Algorithmus: Gemessene Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) und Gesamtkosten $L(N,n)$ (unten links) sowie Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$. Die gemessenen Kosten stimmen mit den erwarteten Kosten überein (vgl. Abb. 4.14).

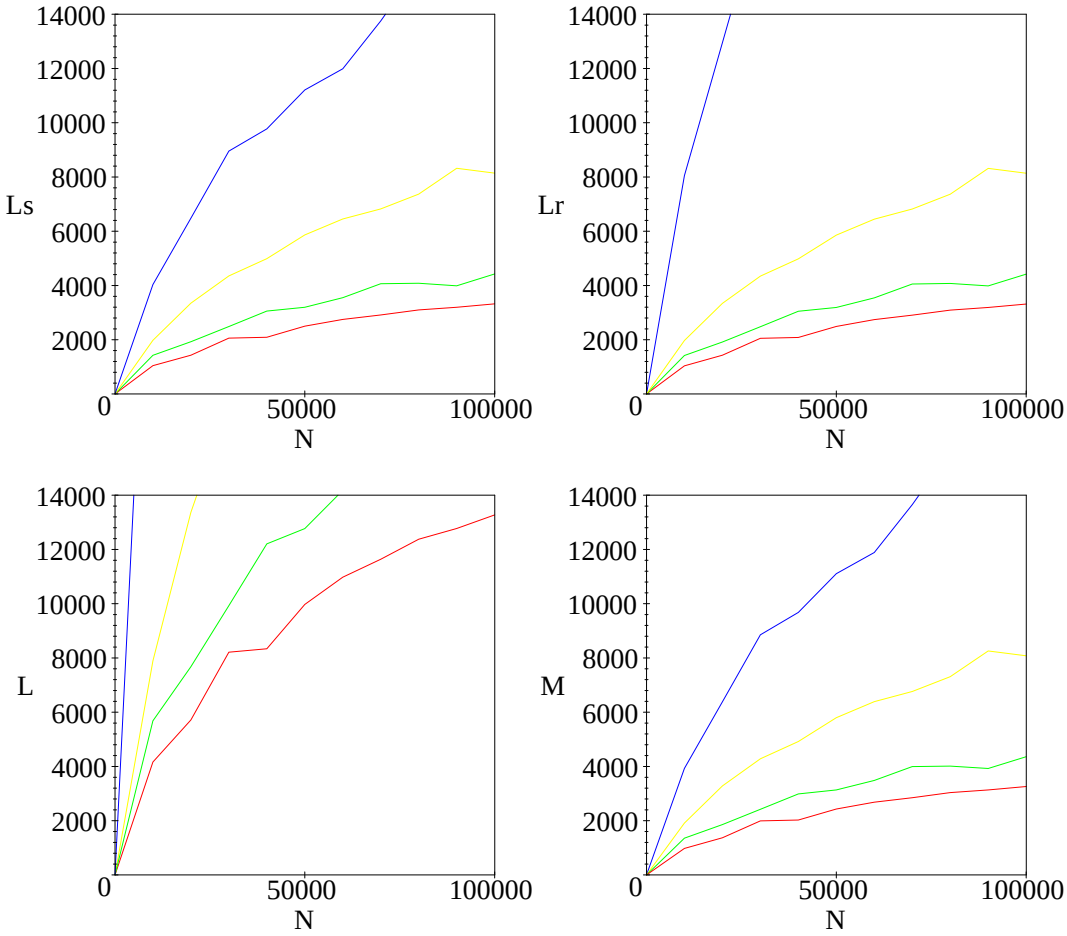


Abbildung 4.32: Fagin-Algorithmus: Gemessene Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) und Gesamtkosten $L(N,n)$ (unten links) sowie Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$. Die gemessenen Kosten stimmen mit den erwarteten Kosten überein (vgl. Abb. 4.16).

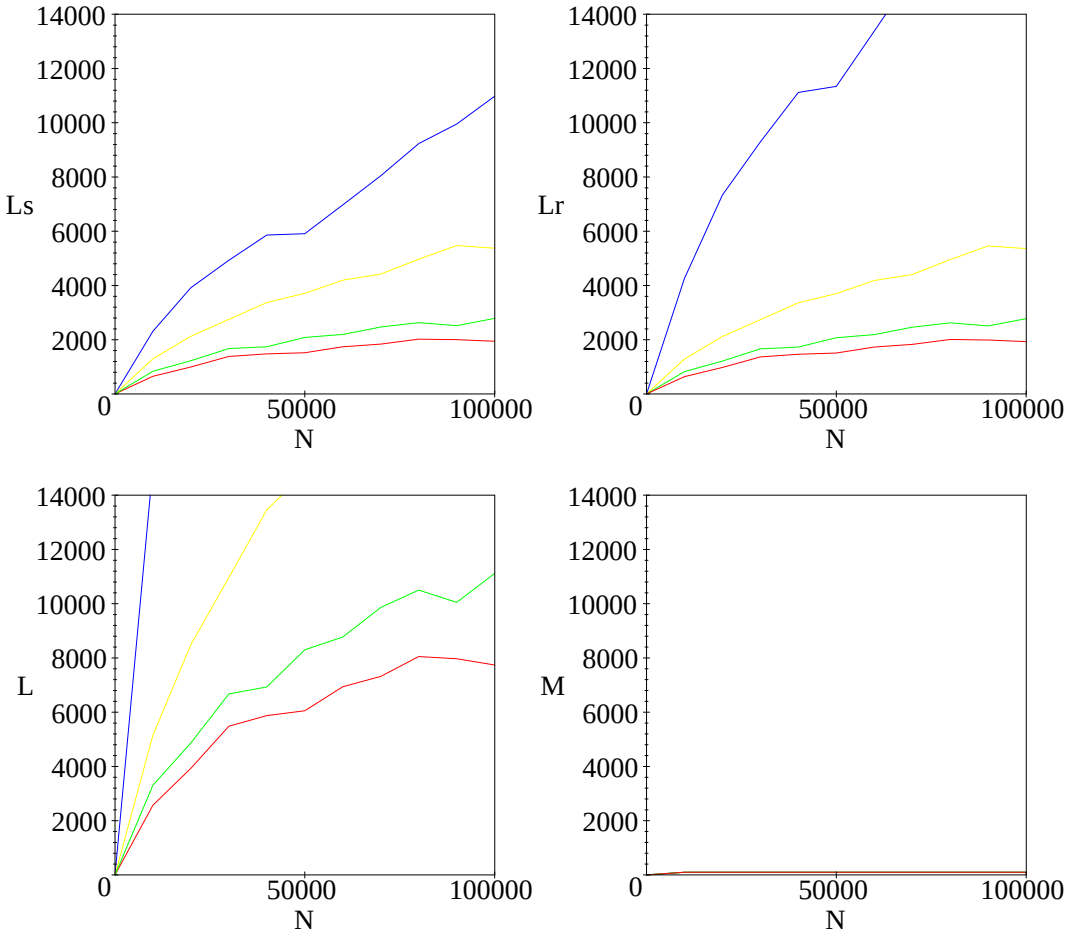


Abbildung 4.33: Threshold-Algorithmus: Gemessene Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) und Gesamtkosten $L(N,n)$ (unten links) sowie Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$. Die gemessenen Kosten stimmen mit den erwarteten Kosten überein (vgl Abb. 4.18).

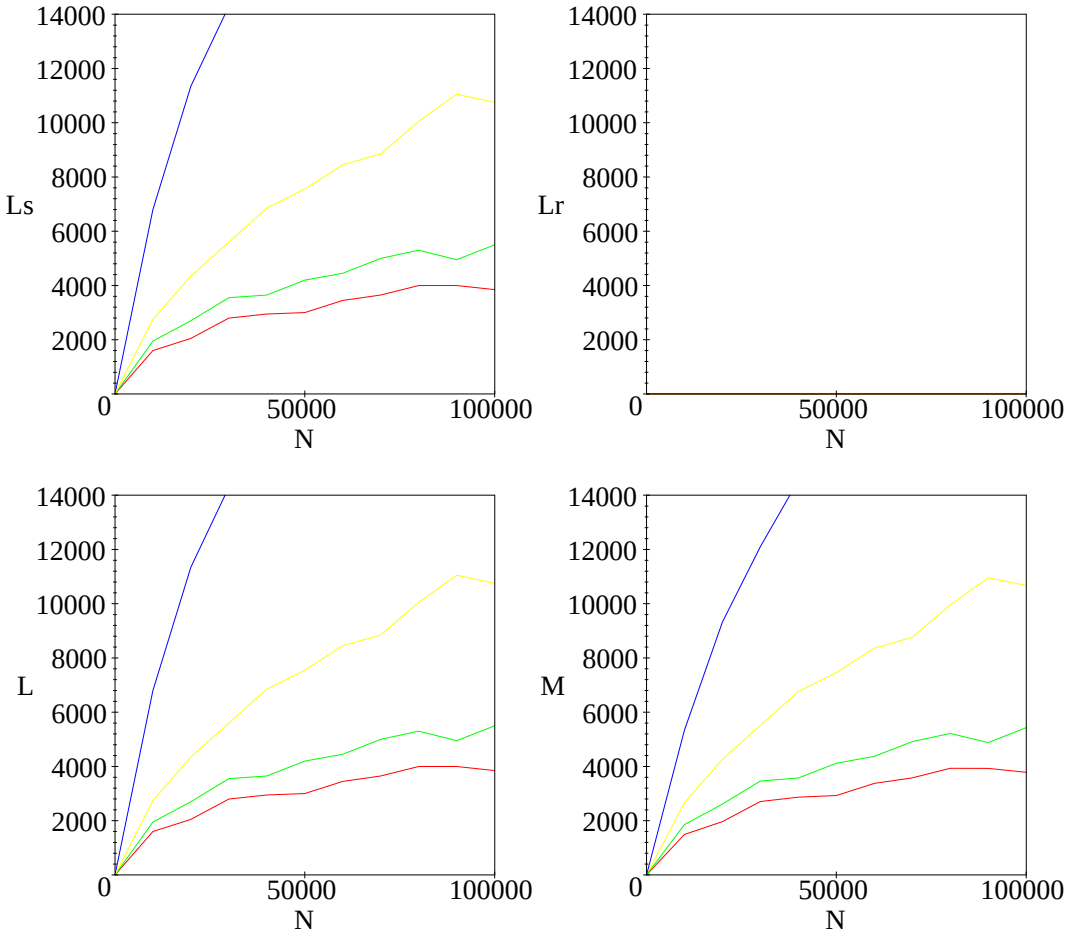


Abbildung 4.34: NRA-Algorithmus: Gemessene Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) und Gesamtlaufzeitkosten $L(N,n)$ (unten links) sowie Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$. Die gemessenen Kosten liegen innerhalb der erwarteten Grenzen (vgl. Abb. 4.21).

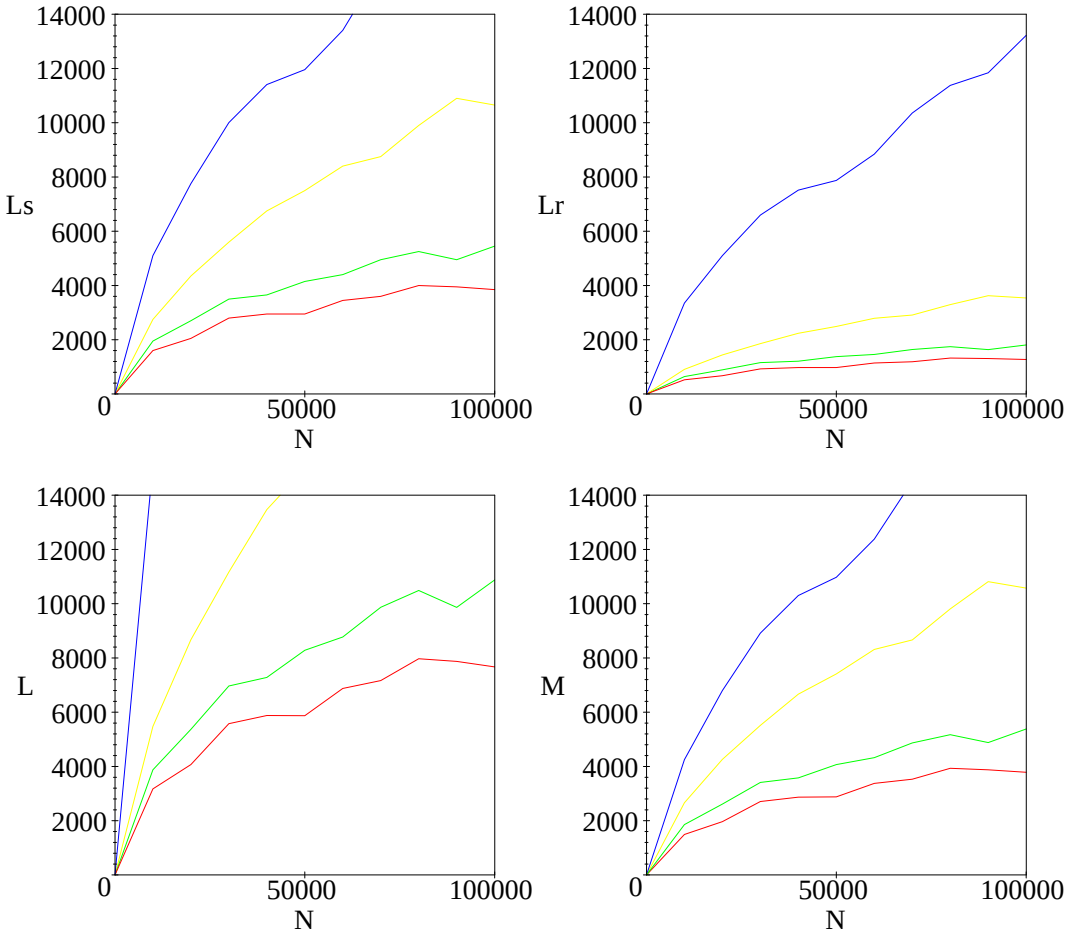


Abbildung 4.35: Combined-Algorithmus: Gemessene Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) und Gesamtkosten $L(N,n)$ (unten links) sowie Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$. Die gemessenen Kosten liegen innerhalb der erwarteten Grenzen (vgl. Abb. 4.22).

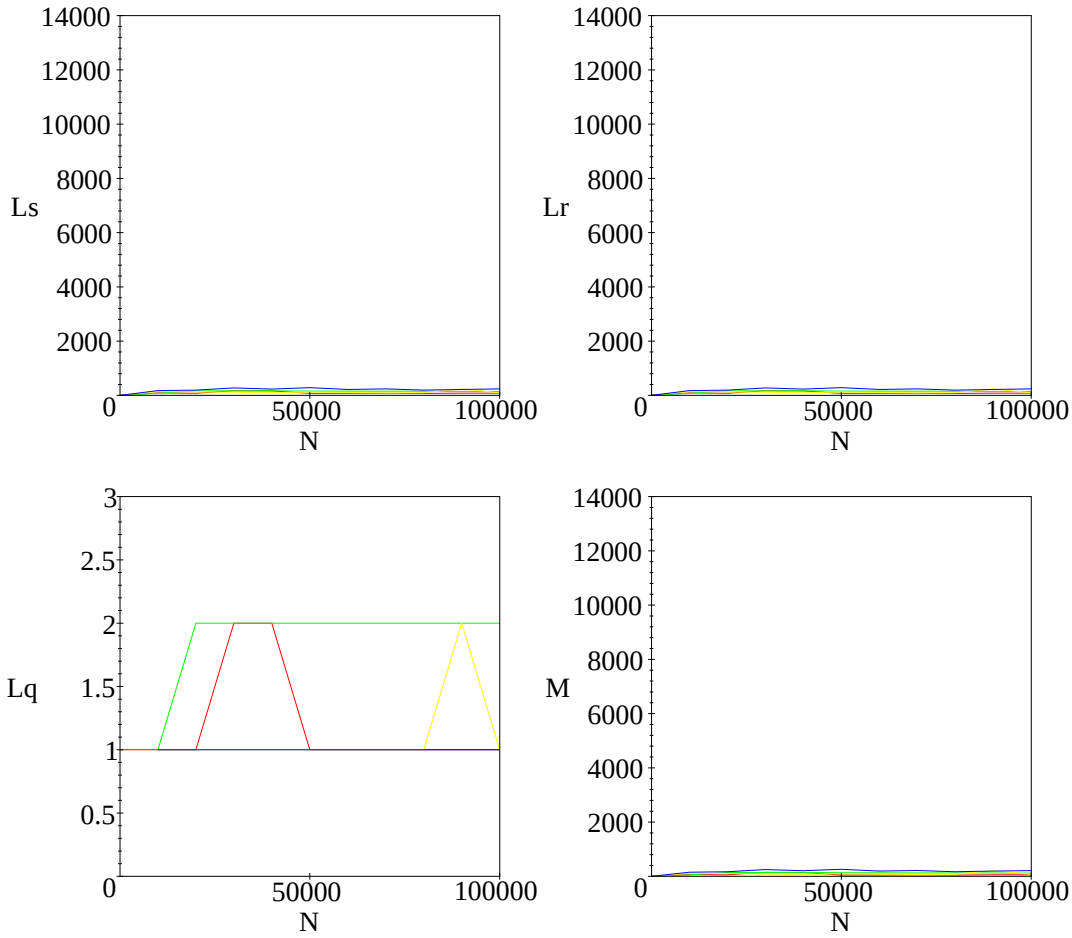


Abbildung 4.36: Discrete-Streams-Algorithmus: Die gemessenen Laufzeitkosten für sorted accesses $L_s(N,n)$ (oben links), random accesses $L_r(N,n)$ (oben rechts) und Quantisierungskosten $L_q(N,n)$ (unten links) sowie Speicherkosten $M(N,n)$ (unten rechts) für Gleichverteilung (rot), Normalverteilung (grün), Exponentialverteilung (gelb) und 3D-Gleichverteilung (blau) für $n = 25$. $L_s(N,n)$ und $L_r(N,n)$ sind erwartungsgemäß minimal und unabhängig von der Größe der Datenbank. $L_q(N,n)$ schwankt aufgrund stochastischer Effekte.

Algorithmus	Laufzeit
Pfeifer-Fuhr-Algorithmus	3.57 sec
Fagin-Algorithmus	7.95 sec
Threshold-Algorithmus	4.99 sec
No-Random-Access-Algorithmus	4.98 sec
Combined-Algorithmus	16.06 sec
Discrete-Streams-Algorithmus	0.75 sec

Tabelle 4.3: Laufzeiten der verschiedenen Algorithmen für die Verknüpfung von 2 gleichverteilten Attributen für $n = 25$ angeforderte Datensätze bei einer Datenbankgröße von $N = 100.000$ unter Verwendung von random accesses.

Algorithmus	Laufzeit
Pfeifer-Fuhr-Algorithmus	6.97 sec
Fagin-Algorithmus	6.39 sec
Threshold-Algorithmus	4.01 sec
No-Random-Access-Algorithmus	8.42 sec
Combined-Algorithmus	17.53 sec
Discrete-Streams-Algorithmus	0.57 sec

Tabelle 4.4: Laufzeiten der verschiedenen Algorithmen für die Verknüpfung von 2 gleichverteilten Attributen für $n = 25$ angeforderte Datensätze bei einer Datenbankgröße von $N = 100.000$ unter Verwendung von direct accesses.

4.4.3 Benutzerbefragung über den Rechercheassistenten

Zur Evaluation des Rechercheassistenten und seiner Fuzzy-Suchsemantik wurde nach etwa $2/3$ der Projektlaufzeit eine Benutzerbefragung durchgeführt. Diese bezieht sich daher auf einen älteren Prototyp des Rechercheassistenten. In der endgültigen Version konnten die Hinweise, die aus der Benutzerbefragung gezogen wurden, genutzt werden, um den Rechercheassistenten weiter zu optimieren.

Die Evaluation wurde parallel auf zwei Wegen durchgeführt: Nach einer Suche mit dem Rechercheassistenten wurde eine Online-Befragung über der Ergebnistabelle eingeblendet, mit der der Benutzer ein Feedback geben konnte. Da mit steigendem Umfang eines solchen Online-Fragebogens die Bereitschaft zur Beteiligung abnimmt, wurde dieser bewusst kurz gehalten.

Um ein detaillierteres Bild über den Zufriedenheitsgrad der Benutzer zu erhalten, wurde eine ausführlichere Fragebogenaktion durchgeführt, an der Studenten unterschiedlichster Semester und Fachrichtungen teilnehmen konnten.

4.4.3.1 Die Online-Befragung

In der Online-Befragung sollten die Benutzer nach Durchführung einer Suche mit dem Rechercheassistenten folgende Aspekte bewerten:

1. die Bedienung der Suchmaske
2. die Qualität des Suchergebnisses
3. die Qualitätssteigerung durch den Rechercheassistenten im Vergleich zu den bisherigen Suchmöglichkeiten (OPAC, Metasuche).

Dazu konnten sie Noten auf einer Skala von 1 bis 5 (”ausgezeichnet”, ”gut”, ”zufriedenstellend”, ”noch ausreichend”, ”mangelhaft”) vergeben. Zusätzlich bestand die Möglichkeit, freie Kommentare abzugeben.

Im Zeitraum der Befragung gingen 131 Bewertungen ein. Aufgrund der Möglichkeit, die vorangegangene Suche zu protokollieren, ergaben sich aufschlussreiche Aspekte bei der statistischen Analyse.

So ging nur 79% der eingegangenen Bewertungen eine korrekte Suche voraus, bei den übrigen 21% hatten sich Tippfehler in den Suchbegriffen eingeschlichen oder die Benutzer kamen mit der Suchmaske nicht klar und gaben z.B. alle Stichworte in eine Zeile ein, also als eine einzige „Such-Phrase”.

Nur 48% aller Benutzer gaben bei ihrer Recherche mehr als 2 Stichworte ein, mehr als die Hälfte verwendete lediglich ein oder zwei Stichworte. Da bei einer derartig kleinen Anzahl an Stichworten nur ein geringer Unterschied zwischen der Fuzzy-Suchlogik und der booleschen Stichwortsuche vorherrscht, wurde die Auswertung der Benutzerfeedbacks in drei Kategorien geteilt:

1. alle Feedbacks
2. alle Feedbacks, denen eine korrekte Suche (ohne Schreib- und Bedienungsfehler) vorausging
3. Feedbacks, denen eine Suche mit mehr als zwei Stichworten vorausging

Wie zu erwarten, weist die Bewertung der einzelnen Fragen eine positive Tendenz von Kategorie 1 zu Kategorie 3 auf, da die Vorteile der Fuzzy-Suche gegenüber der booleschen Suche mit steigender Stichwortanzahl zunehmen.

Für die einzelnen Aspekte zeigten sich folgende Ergebnisse.

Frage 1: Bewertung der Bedienung der Applet-basierten Suchmaske des ersten Rechercheassistenten

Frage 1	Kategorie 1	Kategorie 2	Kategorie 3
1= ”ausgezeichnet”	23,4%	27,7%	28,6%
2= ”gut”	43,8%	49,5%	54,0%
3= ”zufriedenstellend”	14,8%	11,9%	11,1%
4= ”noch ausreichend”	7,8%	5,9%	1,6%
5= ”mangelhaft”	10,2%	5,0%	4,8%
Durchschnittsnote	2,4	2,1	2,0

Die Bedienung der alten Suchmaske wird mit einer Durchschnittsnote zwischen 2.0 und 2.4 bewertet. Diese tendenziell gute Bewertung relativiert sich jedoch durch die eingangs dargelegte Tatsache, dass immerhin 21% der Benutzer Eingabe-Schwierigkeiten hatten. Ein großer Teil dieser Benutzer scheint sich dieser Probleme also gar nicht bewusst zu sein.

In der neuen Version des Rechercheassistenten wurde daher großer Wert auf eine konsequente Vereinfachung der Suchmaske gelegt. Eine zusätzliche „Expertensuchmaske”, die dem Benutzer die gesamte Funktionalität weiterhin offeriert, wurde jedoch beibehalten.

Frage 2: Bewertung der Qualität des Ergebnisses / Frage 3: Bewertung der Qualitätssteigerung im Vergleich mit den bisherigen Suchmöglichkeiten (OPAC, Metasuche)

Frage 2	Kategorie 1	Kategorie 2	Kategorie 3
1= ”ausgezeichnet”	11,3%	14,4%	15,0%
2= ”gut”	25,8%	30,9%	35,0%
3= ”zufriedenstellend”	32,3%	30,9%	33,3%
4= ”noch ausreichend”	11,3%	12,4%	6,7%
5= ”mangelhaft”	19,4%	11,3%	10,0%
Durchschnittsnote	3,0	2,8	2,6

Frage 3	Kategorie 1	Kategorie 2	Kategorie 3
1= ”ausgezeichnet”	18,2%	22,3%	16,9%
2= ”gut”	37,2%	45,7%	55,9%
3= ”zufriedenstellend”	22,3%	19,1%	18,6%
4= ”noch ausreichend”	9,9%	8,5%	6,8%
5= ”mangelhaft”	12,4%	4,3%	1,7%
Durchschnittsnote	2,6	2,3	2,2

Da die Ergebnisqualität und deren Bewertung von der Korrektheit und Adäquatheit der vorangegangenen Suche abhängig ist, zeigen sich hier entsprechende Unterschiede zwischen den drei Benutzer-Kategorien. Insgesamt wurde Qualität mit $2.6 - 3.0$ bewertet, also etwas besser als „zufriedenstellend”. Dieses Ergebnis ist nicht zuletzt auf immerhin 19.4% aller Befragten der Kategorie 1 zurückzuführen, die die Qualität als „mangelhaft” bewerten. Dies entspricht jedoch ziemlich genau dem Prozentsatz der Benutzer, die gravierende Eingabefehler bei ihrer Suche gemacht haben!

Die absolute Mehrheit der Benutzer, die eine korrekte Anfrage gestellt haben (55.9%), gestehen dem Rechercheassistenten eine „gute” Qualitätssteigerung gegenüber dem restlichen Angebot der Bibliothek zu, 16.9% sogar eine „sehr gute”. Dementsprechend wird die Qualitätssteigerung zwischen 2.2 und 2.6, auch insgesamt eher mit „gut” bewertet.

Dass die Bewertung der Qualitäts*steigerung* im Vergleich zur Bewertung der absoluten Qualität deutlich günstiger ausfällt, kann auch auf eine generelle Unzufriedenheit mit Recherchesystemen zurückgeführt werden. In der Benutzer-Kategorie 3 der Benutzer, die eine sinnvolle Anfrage gestellt haben, zeigen sich dann auch die differenziertesten Unterschiede in der Bewertung der Qualität im Vergleich zur Bewertung der Qualitätssteigerung.

Dabei wird die Qualitätssteigerung gut bewertet, obwohl aufgrund der noch nicht optimalen Einbindung der ersten Version des Rechercheassistenten in das Web-Angebot der Universitätsbibliothek viele Funktionen noch nicht zugänglich waren und nur ein Bruchteil der Datenbanken durchsucht werden konnte, die über die herkömmliche Metasuche der Bibliothek erreicht werden.

Insgesamt wurde also der erste Rechercheassistent als positive Alternative zur herkömmlichen Suche begrüßt. Welche Aspekte und Features als durchaus vorhandenes Optimierungspotenzial besonders interessant sind, und welche Änderungen sinnvollerweise an der neuen Version des Rechercheassistenten vorgenommen wurden, zeigt die detailliertere Fragebogenaktion, die im folgenden Kapitel dargestellt wird.

4.4.3.2 Die Fragebogenaktion

Die Fragebogenaktion hatte als Ergänzung zur Online-Befragung einen umfangreicheren Fragen-Katalog; mit insgesamt 14 Fragen wurden 5 Themenbereiche abgedeckt:

1. allgemeine Fragen zum Rechercheassistenten
2. Fragen zur Bedienung der Applet-basierten Suchmaske
3. Fragen zum Suchergebnis bzw. zur angezeigten Trefferliste
4. Fragen zur Gesamtbeurteilung des Rechercheassistenten
5. Fragen zur Person der Befragten (Studienfächer)

Bei „bewertenden" Fragen wurde genau wie bei der Online-Befragung i.d.R. eine fünfteilige Skala verwendet, um einen Vergleich zwischen beiden Evaluations-Arten ziehen zu können. Die übrigen Fragen waren als Multiple-Choice-Fragen formuliert oder boten die Möglichkeit zu freien Kommentaren.

Insgesamt wurden 28 Fragebögen selbständig ausgefüllt, d.h. ohne begleitende Interviews. Das Benutzerfeedback kam von Studenten aus dem Grund- und Hauptstudium, die unterschiedlichste Studiengänge belegten.

Allgemeine Fragen zum Rechercheassistenten

1. Wie sind Sie auf den Rechercheassistenten aufmerksam geworden?

[1]	durch die Homepage der Bibliothek	26,67%
[2]	durch Bibliotheksmitarbeiter	66,67%
[3]	durch sonstige Personen	6,67%

2. Wie oft benutzen Sie den Rechercheassistenten im Verhältnis zur herkömmlichen Suche im OPAC oder mit der Metasuche der Bibliothek?

[1]	selten	36,67%
[2]	Ich benutze den Rechercheassistenten, wenn ich mit den sonstigen Suchinstrumenten nichts finde.	26,67%
[3]	genauso häufig wie sonstige Suchen	10,00%
[4]	Ich suche immer über den Rechercheassistenten.	0,00%
[5]	sonstiges	26,67%

Die Antworten auf Fragen 1 und 2 zeigen, dass der Rechercheassistent im Vergleich zu der herkömmlichen Suche anfangs nur selten genutzt wurde. Die Angaben unter „sonstiges" (ausschließlich „zum ersten Mal") bestätigen dieses Ergebnis.

Die Nutzung ist dabei natürlich stark abhängig von der Art der Einbindung in das Gesamt-Angebot. Da zur Zeit der Befragung noch kein direkter Link von der Startseite der Bibliothek existierte, ist die geringe Nutzung nicht weiter verwunderlich. Wenn mit der herkömmlichen Suche im OPAC keine Treffer gefunden wurden, wurde ein Link zum Rechercheassistenten angezeigt, was den hohen Prozentsatz der Antwort 2 auf Frage 2 erklärt.

Die Ergebnisse dieser Fragen lassen außerdem Rückschlüsse darauf zu, dass sich die Befragten noch nicht lange mit dem Rechercheassistenten auseinander gesetzt haben.

3. Zum Rechercheassistenten werden einige Hilfetexte angeboten (unter ”So geht’s” und ”FAQ”). Wie hilfreich finden Sie dieses Angebot?

[1]	sehr hilfreich	3,57%
[2]		42,86%
[3]		25,00%
[4]		3,57%
[5]	überhaupt nicht hilfreich	0,00%
	Durchschnittsnote: 2,4	
	Ich habe die Hilfetexte bisher nicht gelesen.	25,00%

Zunächst fällt auf, dass ein Viertel der Befragten die Hilfetexte gar nicht zur Kenntnis genommen hat. Da es sich bei den Befragten zum großen Teil um Erstbenutzer handelte, die durch Bibliotheksmitarbeiter auf den Rechercheassistenten aufmerksam gemacht wurden (vgl. Fragen 1 und 2), ist zu erwarten, dass der Durchschnittsbenutzer sich noch weit weniger mit den Texten auseinander setzt.

Von den Benutzern, die die Texte gelesen haben, fanden immerhin knapp 2/3 sie „hilfreich” bis „sehr hilfreich”.

Dies zeigt, dass die Qualität der Hilfetexte sehr hoch ist, und gab gleichzeitig den Hinweis darauf, die Hilfe in der neuen Version des Rechercheassistenten kontextsensitiver zu gestalten und damit gezielter einzusetzen.

Fragen zur Bedienung der Applet-basierten Suchmaske

4. Wie zufrieden sind Sie mit der Verständlichkeit der Bedienung der Suchmaske?

[1]	sehr zufrieden	7,41%
[2]		51,85%
[3]		25,93%
[4]		11,11%
[5]	sehr unzufrieden	3,70%
	Durchschnittsnote: 2,5	

Frage 4 des Fragebogens entspricht Frage 1 der Online-Befragung. Die Applet-basierte Suchmaske des Rechercheassistenten wird mit der Durchschnittsnote von 2.5 auch etwa gleich bewertet wie in der Online-Aktion.

Diese eher durchschnittliche Bewertung wurde zum Anlass genommen, die Suchmaske in der neuen Version komplett zu überarbeiten.

5. Nutzen Sie die nachfolgend genannten Möglichkeiten, die Suchanfrage individuell zu gestalten?

(a)	Einstellung der Aggregationsoperatoren (Auswahl ”möglichst viele”, ”alle”, ”möglichst alle” etc. im Pulldown-Menü unter ”Stichworte/Such-Phrasen”)	
	[ ] Ja	44,44%
	[ ] Nein, ich nutze nur die Voreinstellung.	55,56%
(b)	Einstellung der Relevanz der Suchbegriffe mit Hilfe der Schieberegler (”sehr wichtig”, ”wichtig” etc.)	
	[ ] Ja	74,07%
	[ ] Nein, ich nutze nur die Voreinstellung.	25,93%
(c)	Suche nach Autor	
	[ ] Ja	81,48%
	[ ] Nein	18,52%
(d)	Suche nach Erscheinungsjahr	
	[ ] Ja	52,00%
	[ ] Nein	48,00%

6. Falls Sie einige Fragen unter 5. mit ”nein” beantwortet haben: Warum nutzen Sie nur die jeweilige Voreinstellung bzw. verzichten auf die Zusatzfunktionen der Suche nach Autor oder Erscheinungsjahr?

[1]	Ich habe diese Möglichkeiten bisher nicht bewusst wahrgenommen.	4,55%
[2]	Ich habe bisher keinen Bedarf gesehen, die Voreinstellungen zu verändern.	72,73%
[3]	Ich konnte keine wesentliche Verbesserung der Suchergebnisse durch eine Veränderung der Voreinstellungen feststellen.	4,55%
[4]	sonstiges:	18,18%

Frage 5 und 6 zielten auf eine stärkere Differenzierung der Gesamtbeurteilung der Suchmaske ab. Insgesamt ergibt sich folgendes Bild:

Etwa die Hälfte der Benutzer nutzt die Möglichkeit, den Aggregationsoperator zu verstellen. Untersucht man die Korrelation zwischen den Fragen 5 und 6, so stellt sich heraus, dass die Antwort „Ich habe bisher keinen Bedarf gesehen, die Voreinstellungen zu verändern” sich in 38,5% aller Fälle auf die Aggregationsoperatoren bezieht. Unter diesem Gesichtspunkt ist der Anteil derer, die diese Einstellung verändern, noch als sehr hoch einzuschätzen, zumal das Begreifen der Aggregationsoperatoren den höchsten Grad an Verständnis der Fuzzy-Logik voraussetzt. Gleichzeitig ist objektiv gesehen die Notwendigkeit, den Aggregationsoperator zu verstellen, im Vergleich zu den anderen Suchoptionen am geringsten - nur in wenigen Fällen ist es essentiell, die Precision durch einen restriktiveren Operator zu erhöhen bzw. den Recall durch einen weniger restriktiven Operator zu steigern.

Die Schieberegler zur Veränderung der Suchbegriffs-Gewichte werden von etwa 3/4 der Benutzer verwendet. Diese relativ hohe Nutzungsrate kommt dem Kernpunkt der Fuzzy-Suche

entgegen und zeigt die gute Akzeptanz der Benutzer gegenüber der neuen Suchlogik. Das Herausstellen zentraler Begriffe ihrer Suche und die Kennzeichnung weniger wichtiger Begriffe als „Nebenaspekte" scheint den Nutzern intuitiv verständlich.

Mit 81.48% wird die Autorensuche am häufigsten verwendet, obwohl diese nicht das Primärziel der Fuzzy-Suchlogik darstellt. Aufgrund der Tatsache, dass es sich hierbei - im Gegensatz z.B. zu den Gewichten - um ein bekanntes Suchkriterium handelt, ist die hohe Nutzungsfrequenz jedoch nicht erstaunlich.

Das Erscheinungsjahr wird etwa von der Hälfte der Benutzer als weiteres Suchkriterium herangezogen. Untersucht man Frage 5 und 6 gemeinsam, so zeigt sich wieder, dass sich die Antwort „Ich habe bisher keinen Bedarf gesehen, die Voreinstellungen zu verändern" in 30.8% aller Fälle auf dieses Kriterium bezieht. Da Dokumente, die eine in Bezug auf die Anfrage identische Relevanz aufweisen, in der Ergebnisliste absteigend nach Datum sortiert sind, ist auch hier der hohe Prozentsatz derjenigen, die die zusätzliche Gewichtung nach Datum nicht verwenden, nachvollziehbar.

Letztendlich kann die Frage nach der Wertung, warum einige Suchparameter nur von der Hälfte der Nutzer verändert werden, auf zweierlei Arten beantwortet werden: Entweder ist der Nutzen bzw. der Sinn dieser Parameter den Benutzern nicht verständlich, oder aber die Voreinstellungen sind bereits optimal. Beim Aggregationsoperator, der wie erläutert ein relativ hohes Verständnis der Suchlogik voraussetzt, ist vermutlich die erste Antwort zutreffend. Bei den Gewichten, die ja von sehr vielen Benutzern variiert werden und deren Sinn offensichtlich verstanden wird, trifft auf die 25.93% der Benutzer, die die Gewichte trotzdem nicht verstellen, möglicherweise die zweite Antwort zu: Schließlich sind die Gewichte so voreingestellt, dass der zuerst eingegeben Suchbegriff die maximale Relevanz bekommt, und die nächsten eingegebenen Suchbegriffe ein immer geringeres Gewicht erhalten - so wie dem Benutzer bei seiner Recherche höchstwahrscheinlich die wichtigsten Suchbegriffe zuerst einfallen, und die weniger wichtigen erst im Anschluss daran.

Egal, wie die Frage nach der Bewertung beantwortet wird, unterstreicht das Ergebnis der Fragen 5 und 6 die Einschätzung, die sich auch aus den übrigen Antworten und Anmerkungen der Benutzer ergeben hat: Die alte Suchmaske bot gerade für den Einstig in die Suche zu viele Optionen und wird als „überladen" empfunden. Trotzdem werden alle Parameter zumindest von einigen Benutzern verwendet. Die Trennung der Maske in eine einfache Suche, bei der die optimalen Parameter automatisch gefunden werden und eine Expertensuche, die den vollen Funktionsumfang zur Verfügung stellt, wird durch dieses Ergebnis also unterstützt.

7. Eine besondere Eigenschaft des Rechercheassistenten ist es, dass er Suchen auch mit vielen Suchbegriffen zugleich unterstützt. Halten Sie diese Eigenschaft für vorteilhaft?

[1]	sehr vorteilhaft	37,04%
[2]		48,15%
[3]		11,11%
[4]		3,70%
[5]	überhaupt nicht vorteilhaft	0,00%
	Durchschnittsnote: 1,8	

8. Verwenden Sie bei einer Suche im Rechercheassistenten mehr oder weniger Suchbegriffe als bei einer Suche im OPAC oder in der Metasuche?

	Ich verwende	
[1]	mehr Suchbegriffe	74,07%
[2]	genauso viele Suchbegriffe	22,22%
[3]	weniger Suchbegriffe.	3,70%

Eine große Mehrheit von 3/4 der Benutzer gibt an, mehr Suchbegriffe als in anderen Suchmaschinen zu verwenden und bewertet die Möglichkeit dazu als „vorteilhaft" bzw. „sehr vorteilhaft", was der Fuzzy-Suchlogik entgegenkommt. Trotzdem ist hier eine Diskrepanz zu der Online-Befragung festzustellen, bei der die Hälfte der Benutzer maximal zwei Stichworte verwendeten. Vermutlich ist dies dadurch zu erklären, dass sich die Teilnehmer der Fragebogenaktion intensiver mit dem Rechercheassistenten auseinander gesetzt haben und z.B. auch die Hilfetexte gelesen haben.

Fragen zum Suchergebnis bzw. zur angezeigten Trefferliste

9. Sind Sie mit den Suchergebnissen, die Ihnen der Rechercheassistent liefert, zufrieden?

[1]	sehr zufrieden	14,81%
[2]		37,04%
[3]		37,04%
[4]		11,11%
[5]	sehr unzufrieden	0,00%
	Durchschnittsnote: 2,4	

Vergleichen Sie nun die Qualität der Suchergebnisse im Rechercheassistenten mit der Qualität der Suchergebnisse im OPAC bzw. in der Metasuche und bewerten Sie:
Ich bewerte die Qualität der Suchergebnisse im Rechercheassistenten gegenüber derjenigen der Suchergebnisse im OPAC bzw. in der Metasuche als

[1]	überlegen	53,85%
[2]	gleichwertig	34,62%
[3]	unterlegen.	11,54%

Frage 9 entspricht Fragen 2 und 3 der Online-Befragung. Die Qualität des Suchergebnisses wird hier mit der Note 2.4 etwas besser bewertet als in der Online-Befragung, außerdem gibt es keine „sehr unzufriedene" Bewertung. (Natürlich erleichtert die größere Anonymität in der Online-Befragung die Vergabe schlechterer Noten.)

Immerhin sieht die absolute Mehrheit der Benutzer den Rechercheassistenten als überlegen an, und nur gut jeder zehnte bezeichnet ihn als unterlegen - trotz wie erläutert anfangs geringerem Datenbank-Angebot und schlechterer Einbindung in die Zusatzdienste der Bibliothek. Damit ergibt sich ein fast genauso gutes Bild wie bei der Online-Befragung.

Die Kommentare zu dieser Frage zeigen, dass die Qualitätssteigerung vor allem auf die Möglichkeit, mehrere Suchbegriffe eingeben zu können, zurückzuführen sind. Sowohl die höhere Trefferzahl als auch die höhere Präzision des Rechercheassistenten gegenüber den herkömmlichen Suchwerkzeugen werden positiv hervorgehoben. Das Ziel der Fuzzy-Suchlogik ist damit erreicht.

10. Die Trefferliste im Rechercheassistenten präsentiert die Treffer sortiert nach ihrer Relevanz für die Suchanfrage (Ranking). Entspricht das Ranking Ihrer eigenen Einschätzung der Relevanz der Treffer?

Das Ranking entspricht meiner Relevanzeinschätzung

[1]	sehr gut	3,70%
[2]		40,74%
[3]		40,74%
[4]		7,41%
[5]	überhaupt nicht	7,41%
	Durchschnittsnote: 2,7	

Die allermeisten Benutzer beurteilen das Ranking als „gut" oder „befriedigend", im Durchschnitt ergibt sich eine Gesamtnote von 2.7. Dies kann - v.a. in Verbindung mit der guten Bewertung der vorangegangenen Frage nach der Qualitätssteigerung - als positives Urteil angesehen werden, obwohl natürlich weiteres Optimierungspotential besteht.

Fragen zur Gesamtbeurteilung des Rechercheassistenten

11. Wie beurteilen Sie den Rechercheassistenten insgesamt?

Ich bin mit dem Rechercheassistenten

[1]	sehr zufrieden	7,41%
[2]		55,56%
[3]		29,63%
[4]		7,41%
[5]	sehr unzufrieden	0,00%
	Durchschnittsnote: 2,4	

Die Gesamtbewertung des Rechercheassistenten fällt ausgesprochen positiv aus - eine deutliche Mehrheit von ca. 63% ist „zufrieden” oder sogar „sehr zufrieden”.

Einige Kommentare zu dieser Frage heben wieder die thematische Suche mit mehreren Stichworten - also die Fuzzy-Suchlogik - hervor.

12. Welche Anregungen und Kritikpunkte, die in den bisherigen Fragen nicht berücksichtigt wurden, sind Ihnen noch wichtig?

Die allgemeinen Kritikpunkt und Anregungen zu dem Rechercheassistenten lassen sich in drei Kategorien zusammenfassen:

1. Technische Aspekte: Diese umfassen vor allem Kompatibilitäts-Probleme mit der Applet-basierten Suchmaske. Einige Benutzer beklagen lange Antwortzeiten.
2. Verbesserung der Navigation/Suchmaske: Gewünscht wird eine Vereinfachung der Suchmaske mit weniger und überschaubareren Suchoptionen, sowie eine verbesserte Navigation zwischen Suchmaske, tabellarischer Anzeige und Langanzeige.
3. Integration weiterer Dienste: Die Benutzer wünschen die Einbindung weiterer Dienstleistungen wie Abfrage des Ausleihstatus, die Möglichkeit, das Ergebnis per email zu Versenden und die Integration weiterer Datenbanken. Die Einbindung der JASON-Online-Bestellung wird positiv hervorgehoben.

Diese Anregungen wurden bei der Weiterentwicklung des Rechercheassistenten berücksichtigt und die technischen Probleme behoben.

4.4.3.3 Fazit

Beide Teile der Evaluation ergeben ein einheitliches Bild, welches sich wie folgt zusammenfassen lässt:

- Insgesamt wird der Rechercheassistent positiv bewertet und als willkommene Alternative zur herkömmlichen Suche begrüßt. Wenngleich die Ergebnisqualität selten als „sehr gut” bewertet wird, sind sich die Benutzer einig, dass eine deutliche Steigerung gegenüber herkömmlichen Suchinstrumenten erreicht wurde. Vor allem die neue Suchmethodik, die aufgrund der Fuzzy-Logik mehr Suchbegriffe zulässt, wird gut angenommen.
- Die alte Suchmaske des Rechercheassistenten wird zwar generell positiv, aber als zu komplex wahrgenommen. Viele Möglichkeiten, die Suche zu optimieren werden nicht oder nur unzureichend genutzt, außerdem treten Bedienungsfehler auf.

In der neuen Version des Rechercheassistenten wird dieses Problem umgangen, indem eine Trennung erfolgt in eine einfache Suchmaske und eine Expertensuchmaske. Erstere ist angepasst an herkömmliche Internet-Suchmaschinen und besteht im Wesentlichen aus einer Eingabezeile, die den schnellen und intuitiven Einstieg in die Recherche erlaubt.

Die Expertensuchmaske macht dem fortgeschrittenen Benutzer den vollen Funktionsumfang des Rechercheassistenten zugänglich und erlaubt die individuelle Optimierung der Suche.

Die neuen Suchmasken sind rein HTML-basiert, so dass nicht mehr mit technischen Komplikationen zu rechnen ist.

- Die Online-Hilfe wird zwar als sehr informativ angenommen, jedoch nicht sehr häufig genutzt.

  In der neuen Version werden die Hinweise kontextsensitiv eingeblendet, d.h. in überschaubarem Maße und sehr gezielt.

- Die Ergebnisqualität wird im Wesentlichen als „gut" bewertet, Eingabe- und Bedienungsfehler vermindern die positive Bewertung teilweise in Richtung „zufriedenstellend". Auffällig ist dabei die deutlich bessere Bewertung der „Qualitätssteigerung" gegenüber herkömmlichen Systemen, die auf eine generelle Unzufriedenheit vieler Benutzer mit Informations-Systemen schließen lässt.

  Die neue Version des Rechercheassistenten schließt Fehlbedienungen durch eine stark vereinfachte Suchmaske weitestgehend aus, erkennt Tippfehler (mit denen ein nicht zu vernachlässigender Anteil aller eingegebenen Anfragen behaftet war) und gibt eine deutlich sichtbare Warnung aus. Auch bei Eingabe von zu wenigen Suchbegriffen wird ein entsprechender Hinweis ausgegeben.

- Die Benutzer legen Wert auf die Integration von Zusatzfunktionalitäten in den Rechercheassistenten.

  Dies ist in der neuen Version durch mehrere Features, wie „query expansion" (eine auswählbare Liste von automatisch vorgeschlagenen, zusätzlichen Stichworten), eine Ähnlichkeitssuche zu bereits gefundenen Dokumenten und die Integration der Zusatzfunktionalitäten des Bibliotheks-OPACs (z.B. Überprüfung des Ausleihstatus von gefundenen Titeln oder der Möglichkeit, das Suchergebnis per E-Mail zu versenden) geschehen.

- Die Benutzer wünschen eine bessere Navigation zwischen Suchmaske, Ergebnisliste und Vollanzeige. Auch diese wurde in der neuen Version des Rechercheassistenten entscheidend verbessert. Der Rechercheprozess wird durch die Integration der Suchmaske in die Ergebnisseite, sowie durch Tipps zur Optimierung der Suche und durch nützliche Zusatzfunktionen und Navigationshilfen erleichtert.

4.5 Zusammenfassung

In diesem Kapitel wurde die effiziente Aggregation von Fuzzy-Mengen auf Basis von Streaming-Algorithmen untersucht. Zunächst wurden als Kostenmaße die sequentiellen Zugriffe auf die Streams (sorted accesses) und die direkten Datenbankzugriffen (random accesses), sowie die Kosten für die Ausführung einer Datenbankanfrage (bzw. Quantisierungskosten) ausgemacht. Im Anschluss wurden die entscheidenden Einflussfaktoren auf diese Größen ermittelt. Neben den offensichtlichen Faktoren, wie Datenbankgröße, Anzahl angeforderter Ergebnisse und Spezifika der zugrundeliegenden Datenbanken, stellte sich insbesondere die statistische Verteilung der Attribute als erheblicher Einflussfaktor heraus.

Darauf aufbauend wurden verschiedene aus der Literatur bekannte Algorithmen vorgestellt und eingehend untersucht. Des weiteren wurde ein Verfahren, der Discrete-Streams-Algorithmus, speziell für die Fuzzy-Stichwortsuche entwickelt, der aber auch im allgemeinen Fuzzy-Retrieval sehr gute Ergebnisse erzielt.

Als Testszenarien für praktische Messungen und zur Verifikation der theoretisch hergeleiteten Ergebnisse dienten mit dem „Rechercheassistenten" die Stichwortsuche und eine umfangreiche exemplarische SQL-Datenbank, die unter kontrollierten Bedingungen verschiedenen Konfigurationen unterzogen werden konnte.

Der Rechercheassistent, der als praktische Anwendung im Betrieb ist, wurde außerdem einer Benutzerbefragung unterzogen, die die Akzeptanz der eingesetzten Fuzzy-Suchsemantik bewies.

Zusammenfassend lässt sich feststellen:

- Der Pfeifer-Fuhr-Algorithmus ist - obwohl der einfachste und naheliegendste der Algorithmen - erstaunlich effizient und der beste Algorithmus, falls die Anwendung engen Restriktionen unterliegt und lediglich sorted accesses, aber keine random accesses und keine Quantisierung möglich sind.
- Der Fagin-Algorithmus ist in jedem der Kostenmaße schlechter als der Threshold-Algorithmus.
- Der Threshold-Algorithmus ist - obwohl weitaus weniger komplex als der NRA oder CA - der effizienteste Algorithmus, wenn die Quantisierungskosten sehr hoch sind und random accesses möglich und vergleichsweise billig sind, insbesondere also bei direct accesses, wenn alle Attribute aus einer Datenquelle stammen und beim Auslesen direkt verfügbar sind.
- Beim NRA und beim CA lohnen sich der deutlich erhöhte kalkulatorische und implementatorische Aufwand gegenüber dem Pfeifer-Fuhr-Algorithmus bzw. dem TA nicht.
- Der Discrete-Streams-Algorithmus ist der mit Abstand effizienteste Algorithmus, stellt jedoch auch die höchsten Anforderungen. Er benötigt die Möglichkeit der Quantisierung eines Attributes und erfordert für optimale Effizienz Vorwissen über die statisti-

sche Verteilung der Attribute. Besonders geeignet ist er, falls die Kosten für die Ausführung einer Anfrage im Vergleich zu den Kosten für das Einlesen vieler Datensätze relativ klein sind (wie z.B. bei der Internet-Metasuche).

5

Diskussion und Ausblick

5.1 Diskussion

Die vorliegende Arbeit hat die verschiedenen Problemstellungen der Fuzzy-Metasuche analysiert, bekannte Lösungsansätze vorgestellt, eigene Ansätze herausgearbeitet und diese einem Vergleich unterzogen. In zwei Anwendungsszenarien wurden die Ansätze auf ihre Praxistauglichkeit getestet. Das mit dieser Arbeit verbundene Projekt „Rechercheassistent" läuft auch zum aktuellen Zeitpunkt, also nach dem Projekt-Ende, im regulären Betrieb der Bibliothek der Universität Bielefeld weiter und erfreut sich mit gut 5.600 Anfragen pro Monat - Tendenz steigend - reger Benutzung.

In der Arbeit wurden die strukturellen Voraussetzungen für eine Fuzzy-Metasuchmaschine erörtert, die durch eine 3-Schichten-Architektur gelegt werden, bei der die Anbindung verschiedener Datenbanken durch kompakte Wrapper erfolgt. Diese ermöglichen eine einheitliche Kommunikation der eigentlichen Suchmaschine mit den zugrundeliegenden Datenbanken, indem sie die Suchanfragen in die Datenbank-spezifische Anfragesyntax übersetzen, und umgekehrt die von der Datenbank gelieferten Ergebnisse in ein einheitliches Ausgabeformat bringen.

Grundlage für die Bewältigung der großen Datenvolumina bildete das Streaming-Prinzip. Auf diesem basierte auch die Darstellung der diversen, von den Wrappern auszuführenden Nachbearbeitungsschritte, die sogar die Integration von Datenbanken, die in ihrem funktionalen Umfang stark eingeschränkt sind, ermöglichen. Es wurde gezeigt, dass eine unvollständige Sortierung der Datenbank und die fehlende Unterstützung von Negationen und Disjunktionen in der Anfragesprache kein prinzipielles Problem für die Einbindung darstellen

muss. Als Grenze für Nachbearbeitungsschritte wurde die Nachbildung von Konjunktionen gefunden, die nicht mit akzeptablem Antwortzeitverhalten implementiert werden kann.

Der zweite Teil der Arbeit behandelte die Formalisierung einer geeigneten Suchsemantik. Es wurden plausible, formale Kriterien für Fuzzy-Aggregations- und Filter-Operatoren aufgestellt, aus denen Anfragen an eine Suchmaschine konstruiert werden können. Diverse aus der Literatur bekannte Operatoren und Theorien (t-, s-Normen, Generalized Quantifiers, FLQ, Σ-/FE-Count, OWA-Operatoren, p-Norm) wurden auf diese Kriterien hin überprüft. Von den untersuchten Operatoren konnte alleine der WAO-Operator, eine Erweiterung der p-Norm, allen Kriterien genügen. Er bot darüber hinaus die Möglichkeit formaler Anfrageoptimierungen, die abschließend diskutiert wurden. In einer Benutzerbefragung wurde eine hohe Akzeptanz dieser Fuzzy-Suchlogik festgestellt.

Das Hauptproblem bei jeder gewichteten Suchmethodik ist jedoch immer der - gegenüber der herkömmlichen, booleschen Suche - erhöhte Suchaufwand und damit die Performanz. Der Hauptteil der Arbeit hat sich daher der effizienten Aggregation von Fuzzy-Streams gewidmet, die durch eingangs dargestellte Quantoren verknüpft wurden. Es wurden zunächst die entscheidenden Einflussfaktoren untersucht. Neben den naheliegenden Faktoren wie der Datenbankgröße, der Anzahl auszugebender Datensätze und der Zahl zu verknüpfender Streams hat sich auch die stochastische Verteilung der Attribute als entscheidender Faktor erwiesen. Es wurden dann praxisorientierte Kostenmaße aufgestellt, wobei inbesondere die benötigten Datenbankzugriffe - differenziert nach sequentiellen Zugriffen auf die Streams (sorted accesses) und direkten Datenbankzugriffen (random accesses/direct accesses) - aber auch der Speicherverbrauch relevant waren.

Auf diesen Grundlagen aufbauend wurden dann bekannte Verfahren zur effizienten Aggregation von Fuzzy-Streams vorgestellt und ein eigenes Verfahren - der Discrete-Streams-Algorithmus - entwickelt. Diese Verfahren wurden auf die Kostenmaße hin untersucht. Die in den praktischen Anwendungsszenarien durchgeführten Messungen bestätigen die Ergebnisse der theoretischen Untersuchung.

Der Discrete-Streams-Algorithmus, der ursprünglich speziell für die Fuzzy-Stichwortsuche entwickelt wurde, hat sich dabei auch in der allgemeinen Fuzzy-Suche in vielfacher Weise ausgezeichnet: Die Laufzeit ist unter einem breiten Spektrum der Parameter minimal und insbesondere nahezu unabhängig von der Größe der Datenbank, wohingegen die Laufzeit der konventionellen Verfahren bei zwei zu verknüpfenden Streams mit der Wurzel der Datenbankgröße ansteigt. Der Speicherverbrauch des Discrete-Streams-Algorithmus ist äußerst gering und unter bestimmten Bedingungen, wie sie z.B. beim bibliographischen Retrieval vorliegen, sogar minimal.

5.2 Ausblick

Die gewichtete Suchmethodik wird auch in Zukunft gegenüber der booleschen Suchlogik noch weiter an Bedeutung gewinnen und diese - zumindest in ihrer benutzerseitigen Anwendung - immer mehr verdrängen. Die durchsuchbaren Datenbestände werden weiter

wachsen und es zeichnet sich ab, dass ganz neue Domänen wie Multimedia-Datenbanken größeren Einfluss zu verzeichnen haben. Die Anforderungen an die Suchmethodik - insbesondere an die Effizienz der Suche in großen Datenbeständen - werden dabei weiter steigen.

Das Aufstellen sinnvoller Kriterien für die Definition geeigneter Fuzzy-Operatoren - insbesondere der Zugehörigkeitsfunktionen - die eine adäquate Suche ermöglichen, ist in gewissem Umfang immer anwendungsspezifisch und daher nur bedingt auf andere Domänen übertragbar.

Die in dieser Arbeit vorgestellten Algorithmen zur effizienten Anwendung dieser Operatoren auf Datenbanken sind jedoch allgemein genug gefasst, um prinzipiell beliebige Suchdomänen und Paradigmen zu behandeln.

Abhängig von den Anforderungen, die an die zu durchsuchenden Datenbanken und zugrundeliegende Infrastruktur gestellt werden dürfen, haben sich im Rahmen dieser Arbeit verschiedene der Algorithmen als günstig erwiesen. So wie sich diese Faktoren relativ zu einander entwickeln werden, können auch in Zukunft verschiedene der Algorithmen erfolgversprechend sein.

Ein entscheidender Einflussfaktor auf die Effizienz der Algorithmen war die Anzahl an zu verknüpfenden Attributen. Die meisten Streaming-Algorithmen skalieren relativ schlecht mit dieser Größe. Bei anderen Algorithmen hingegen war dieser Faktor eher unerheblich, dafür kamen zusätzliche Quantisierungskosten hinzu. Unklar ist, ob diese Quantisierungskosten ähnlich schlecht skalieren (die Messungen bei der Steigerung von 2 auf 3 zu verknüpfende Attribute deuten nicht darauf hin). Interessant wären Untersuchungen über die Aggregation von deutlich mehr als 2 oder 3 Attributen.

Wichtig für die optimale Quantisierung bei dem in dieser Arbeit entwickelten Discrete--Streams-Algorithmus war die Möglichkeit, die stochastische Verteilung der Attribute schätzen zu können. Diese Parameter auch bei unbekannten Datenbanken mit (aufgrund des Aspektes der Metasuche) möglicherweise eingeschränktem Zugriff automatisch und möglichst exakt zu schätzen, ist eine neue Herausforderung. Möglicherweise wäre es - gerade bei pathologischen Verteilungen - lohnenswert, mit einer non-uniformen Quantisierung, d.h. einer Einteilung in nicht gleich große Intervalle zu arbeiten.

Denkbar wäre weiterhin ein Hybrid aus dem Discrete-Streams-Algorithmus und z.B. Pfeifers oder dem Threshold-Algorithmus. Bei mehr als zwei zu verknüpfenden Attributen wäre es z.B. möglich, einen Teil der Attribute zu quantisieren und den Rest unter Ausnutzung von direct accesses mit dem Threshold-Algorithmus zu aggregieren. Speziell bei der Stichwortsuche mit ihren diskreten Zugehörigkeitsgraden wäre es z.B. möglich, anstatt nach allen gewünschten Stichworten zu suchen, zunächst nur einen Teil der Stichworte zu berücksichtigen, und die Relevanz im Nachhinein durch direct accesses entsprechend der restlichen Stichworte anzupassen. Da sich die quantisierten Bereiche dann jedoch unter Umständen überschneiden - eine grobere Granulierung der Relevanz also gar nicht bewirkt, dass Stichworte aus der Suche herausfallen können - erfordert dies eine nicht triviale Anpassung der Algorithmen.

Diese Arbeit liefert durch den einheitlichen Vergleich der Algorithmen untereinander Entscheidungskriterien für die Wahl des unter den jeweiligen Bedingungen günstigsten Verfahrens. Mit dem entwickelten Discrete-Streams-Algorithmus, der speziell für die Fuzzy-Stichwortsuche optimiert wurde, aber nachgewiesenermaßen ebenfalls in anderen Anwendungsszenarien sehr gute Ergebnisse erzielt, liefert sie darüber hinaus die Grundlage für derartige Erweiterungen.

A

Notationen

Die folgenden Tabellen führen die in dieser Arbeit verwendeten Notationen und Symbole auf:

Fuzzy-Mengen und Zugehörigkeitsgrade (Kapitel 3)	
μ	Zugehörigkeitsgrad (allgemein)
$\mu_{F_i}(a)$	Zugehörigkeitsgrad eines Elementes a zu einer Fuzzy-Menge F_i. Falls eindeutig, wird auch kurz geschrieben: μ_i
$\mu_{agg}(\mu_1, ..., \mu_n)$	Zugehörigkeitsgrad eines Elementes zu der Aggregation der Fuzzy-Mengen $F_1...F_n$ mittels des Operators agg, wobei das Element die Zugehörigkeitsgrade μ_i zu den Fuzzy-Mengen F_i aufweist (i=1..n)
$\mu_{agg}(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle)$	Zugehörigkeitsgrad eines Elementes zu der Aggregation der Fuzzy-Mengen $F_1...F_n$ mittels des gewichteten Operators agg, wobei das Element die Zugehörigkeitsgrade μ_i zu den Fuzzy-Mengen F_i aufweist (i=1..n) und mit w_i gewichtet wird.

Verteilungsdichten und Integrale (Kapitel 4)	
$g(x, y)$, $g(x, y, z)$	Verteilungsdichte von Datensätzen über die Attribute x,y(,z)
$G_x(a)$	Anteil der von Stream x eingelesenen Datensätze bei einer Relevanz von a
$G(a)$	Summe der $G_*(a)$, Anteil der von allen Streams eingelesenen Datensätze bei einer Relevanz von a
$M(a)$	Anteil der „matches", also der Datensätze, die bereits von allen Streams gelesen wurden, bei einer Relevanz von a
$A(a)$	Anteil der auszugebenden Datensätze bei einer Relevanz von a

Laufzeitkosten (Kapitel 4)	
c_s	Kosten für einen sequentiellen Zugriff auf einen Stream (sorted access), gemessen in sec
c_r	Kosten für einen wahlfreien Zugriff auf die Datenbank (random access), gemessen in sec
c_q	Kosten für die Ausführung einer Anfrage bzw. Quantisierungskosten, gemessen in sec
h	Verhältnis der Kosten für einen random access zu einem sorted access, $h = c_r/c_s$
$L_s(.)$	Anzahl an sorted accesses in Abh. verschiedener Parameter, insbesondere Datenbankgröße N und Anzahl angeforderter Datensätze n
$L_r(.)$	Anzahl an random accesses in Abh. verschiedener Parameter, insbesondere Datenbankgröße N und Anzahl angeforderter Datensätze n
$L_q(.)$	Anzahl an auszuführenden Datenbankanfragen in Abh. verschiedener Parameter, insbesondere Datenbankgröße N und Anzahl angeforderter Datensätze n
$L(.)$	Gesamt-Laufzeitkosten eines Algorithmus
$\underline{L}(.)$	untere Schranke bzw. Abschätzung der Größe $L(.)$
$\overline{L}(.)$	obere Schranke bzw. Abschätzung der Größe $L(.)$

B

Beweise

B.1 Partielle Distributivität

Aggregations-Operatoren erfüllen die partielle Distributivität nach Def. 3.2.

Beweis:

Da μ_c boolesch, genügt zu zeigen, dass:

$$\mathfrak{s}(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), 0) = \mu_{agg}(\langle w_a, \mathfrak{s}(\mu_a, 0)\rangle, \langle w_b, \mathfrak{s}(\mu_b, 0)\rangle) \quad (1)$$

$$\wedge \quad \mathfrak{s}(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), 1) = \mu_{agg}(\langle w_a, \mathfrak{s}(\mu_a, 1)\rangle, \langle w_b, \mathfrak{s}(\mu_b, 1)\rangle) \quad (2)$$

(1) für t-Norm:

$t(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), 0)$

$= 0$ (Grenzbedingung t-Norm Def. 3.15 (7))

$= \mu_{agg}(\langle w_a, 0\rangle, \langle w_b, 0\rangle)$ (Grenzbedingung Agg.-Op. Def. 3.15 (2))

$= \mu_{agg}(\langle w_a, t(\mu_a, 0)\rangle, \langle w_b, t(\mu_b, 0)\rangle)$ (Grenzbedingung t-Norm Def. 3.15 (7))

(2) für t-Norm:

$t(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), 1)$

$= \mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle)$ (Grenzbedingung t-Norm Def. 3.15 (5))

$= \mu_{agg}(\langle w_a, t(\mu_a, 1)\rangle, \langle w_b, t(\mu_b, 1)\rangle)$ (Grenzbedingung t-Norm Def. 3.15 (5))

(1) für s-Norm:

$s(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), 0)$

$= \mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle)$ (Grenzbedingung s-Norm Def. 3.15 (4))

$= \mu_{agg}(\langle w_a, s(\mu_a, 0)\rangle, \langle w_b, s(\mu_b, 0)\rangle)$ (Grenzbedingung s-Norm Def. 3.15 (4))

(2) für s-Norm:

$s(\mu_{agg}(\langle w_a, \mu_a\rangle, \langle w_b, \mu_b\rangle), 1)$

$= 1$ (Grenzbedingung s-Norm Def. 3.15 (6))

$= \mu_{agg}(\langle w_a, 1\rangle, \langle w_b, 1\rangle)$ (Grenzbedingung Agg.-Op. Def. 3.15 (1))

$= \mu_{agg}(\langle w_a, s(\mu_a, 1)\rangle, \langle w_b, s(\mu_b, 1)\rangle)$ (Grenzbedingung s-Norm Def. 3.15 (6))

B.2 Parametrisierungs-Eigenschaft des ΣCount

Der ΣCount erfüllt für monotone Aggregationsoperatoren *nicht* die Parametrisierungseigenschaft. Beweis durch Widerspruch.

Beweis:

Parametrisierungs-Bedingung:

$$\exists\lambda_1, \lambda_2 \forall w: \quad \mu_{agg\lambda_1}(\langle w, 1\rangle, \langle \tfrac{w}{n}, 0\rangle, ..., \langle \tfrac{w}{n}, 0\rangle) < \mu_{agg\lambda_1}(\langle w, 0\rangle, \langle \tfrac{w}{n}, 1\rangle, ..., \langle \tfrac{w}{n}, 1\rangle)$$
$$\wedge$$
$$\mu_{agg\lambda_2}(\langle w, 1\rangle, \langle \tfrac{w}{n}, 0\rangle, ..., \langle \tfrac{w}{n}, 0\rangle) > \mu_{agg\lambda_2}(\langle w, 0\rangle, \langle \tfrac{w}{n}, 1\rangle, ..., \langle \tfrac{w}{n}, 1\rangle)$$

hier: $W := [w, \frac{w}{n}, ..., \frac{w}{n}], \quad D_1 := [1, 0, ..., 0], \quad D_2 := [0, 1, ..., 1]$

$$\Rightarrow \quad \exists\lambda_1, \lambda_2 \forall w: \quad \mu_{agg\lambda_1}(\Sigma Count(W|D_1)) < \mu_{agg\lambda_1}(\Sigma Count(W|D_2))$$
$$\wedge$$
$$\mu_{agg\lambda_2}(\Sigma Count(W|D_1)) > \mu_{agg\lambda_2}(\Sigma Count(W|D_2))$$

da $\mu_{agg\lambda_{1,2}}$ monoton, folgt:

$$\Rightarrow \quad \exists\lambda_1, \lambda_2 \forall w: \quad \Sigma Count(W|D_1) < \Sigma Count(W|D_2)$$
$$\wedge$$
$$\Sigma Count(W|D_1) > \Sigma Count(W|D_2)$$

Widerspruch!

B.3 Dualität von $ODER_p$ und UND_p

Die P-Norm erfüllt *nicht* die in Kapitel 3.3.2.5 erläuterte „Dualität": Beweis durch Widerspruch.

Beweis:

Bedingung der Dualität:

$$AND_p(\langle w,1\rangle,\underbrace{\langle\frac{w}{n},0\rangle,...,\langle\frac{w}{n},0\rangle}_{n-mal}) < AND_p(\langle w,0\rangle,\underbrace{\langle\frac{w}{n},1\rangle,...,\langle\frac{w}{n},1\rangle}_{n-mal})$$

$$\wedge \quad OR_p(\langle w,1\rangle,\underbrace{\langle\frac{w}{n},0\rangle,...,\langle\frac{w}{n},0\rangle}_{n-mal}) > OR_p(\langle w,0\rangle,\underbrace{\langle\frac{w}{n},1\rangle,...,\langle\frac{w}{n},1\rangle}_{n-mal})$$

$$1-\left(\frac{(w*(1-1))^p+n*(\frac{w}{n}*(1-0))^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}} < 1-\left(\frac{(w*(1-0))^p+n*(\frac{w}{n}*(1-1))^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}}$$

$$\wedge \quad \left(\frac{(w*1)^p+n*(\frac{w}{n}*0)^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}} > \left(\frac{(w*0)^p+n*(\frac{w}{n}*1)^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}}$$

$$-\left(\frac{n*(\frac{w}{n})^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}} < -\left(\frac{w^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}}$$

$$\wedge \quad \left(\frac{w^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}} > \left(\frac{n*(\frac{w}{n})^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}}$$

$$\left(\frac{w^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}} < \left(\frac{n*(\frac{w}{n})^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}}$$

$$\wedge \quad \left(\frac{w^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}} > \left(\frac{n*(\frac{w}{n})^p}{w^p+n*(\frac{w}{n})^p}\right)^{\frac{1}{p}}$$

Widerspruch!

B.4 Monotonie des AND_p-Operators

Der AND_p-Operator ist streng monoton bezüglich der Gewichte:

Beweis:

Strenge Monotonie bezüglich der Gewichte (für $w_n < w'_n$):

$$AND_p(\langle w_1,\mu_1\rangle,...,\langle w_n,\mu_n\rangle) < AND_p(\langle w_1,\mu_1\rangle,...,\langle w'_n,\mu_n\rangle)$$

$$\Leftrightarrow \quad 1-\left(\frac{\sum_{i=1}^{n-1}(w_i*(1-\mu_i))^p+(w_n*(1-\mu_n))^p}{\sum_{i=1}^{n-1}w_i^p+w_n^p}\right)^{\frac{1}{p}} < 1-\left(\frac{\sum_{i=1}^{n-1}(w_i*(1-\mu_i))^p+(w'_n*(1-\mu_n))^p}{\sum_{i=1}^{n-1}w_i^p+w_n'^p}\right)^{\frac{1}{p}}$$

$$\textit{Substitution}: \quad c_1 := \sum_{i=1}^{n-1}(w_i * (1-\mu_i))^p \qquad c_2 := \sum_{i=1}^{n-1} w_i^p$$

$$\Leftrightarrow \quad \left(\frac{c_1+(w_n*(1-\mu_n))^p}{c_2+w_n^p}\right)^{\frac{1}{p}} > \left(\frac{c_1+(w'_n*(1-\mu_n))^p}{c_2+w_n'^p}\right)^{\frac{1}{p}}$$

$$\Leftrightarrow \quad (c_1+(w_n*(1-\mu_n))^p)*(c_2+w_n'^p) > (c_1+(w'_n*(1-\mu_n))^p)*(c_2+w_n^p)$$

$$\Leftrightarrow \quad c_1c_2 + c_1w_n'^p + w_n^p(1-\mu_n)^pc_2 + w_n^p(1-\mu_n)^pw_n'^p > c_1c_2 + c_1w_n^p + w_n'^p(1-\mu_n)^pc_2 + w_n'^p(1-\mu_n)^pw_n^p$$

$$\Leftrightarrow \quad c_1w_n'^p + w_n^p(1-\mu_n)^pc_2 > c_1w_n^p + w_n'^p(1-\mu_n)^pc_2$$

$$\Leftrightarrow \quad c_1w_n'^p - w_n'^p(1-\mu_n)^pc_2 > c_1w_n^p - w_n^p(1-\mu_n)^pc_2$$

$$\Leftrightarrow \quad w_n'^p * (c_1-(1-\mu_n)^pc_2) > w_n^p * (c_1-(1-\mu_n)^pc_2)$$

$$\Leftrightarrow \quad c_1-(1-\mu_n)^pc_2 > 0$$

$$\Leftrightarrow \quad \frac{c_1}{c_2} > (1-\mu_n)^p$$

$$\Leftrightarrow \quad 1-\left(\frac{c_1}{c_2}\right)^{\frac{1}{p}} < \mu_n$$

$$\Leftrightarrow \quad 1-\left(\frac{\sum_{i=1}^{n-1}(w_i*(1-\mu_i))^p}{\sum_{i=1}^{n-1}w_i^p}\right)^{\frac{1}{p}} < \mu_n$$

$$\Leftrightarrow \quad AND_p(\langle w_1,\mu_1\rangle,...,\langle w_{n-1},\mu_{n-1}\rangle) < \mu_n$$

Dies ist die in Def. 3.12 geforderte Voraussetzung. Analog ergibt sich für

$AND_p(\langle w_1,\mu_1\rangle,...,\langle w_n,\mu_n\rangle) > AND_p(\langle w_1,\mu_1\rangle,...,\langle w'_n,\mu_n\rangle)$

die geforderte Voraussetzung

$AND_p(\langle w_1,\mu_1\rangle,...,\langle w_{n-1},\mu_{n-1}\rangle) > \mu_n$.

B.5 Korrelierte Monotonie des AND_p-Operators

Der AND_p-Operator ist *nicht* k-monoton. Beweis durch Widerspruch.

Beweis:

Bedingung der korrelierten Monotonie (für $\mu_k < \mu_l$ und $\forall \epsilon > 0$):

$$AND_p(..., \langle w_k, \mu_k\rangle, \langle w_l, \mu_l\rangle, ...) \leq AND_p(..., \langle w_k - \epsilon, \mu_k\rangle, \langle w_l + \epsilon, \mu_l\rangle, ...)$$

$$\Leftrightarrow \quad 1 - \left(\frac{\sum_{i\neq k,l}(w_i*(1-\mu_i))^p + (w_k*(1-\mu_k))^p + (w_l*(1-\mu_l))^p}{\sum_{i\neq k,l} w_i^p + w_k^p + w_l^p}\right)^{\frac{1}{p}}$$
$$\leq 1 - \left(\frac{\sum_{i\neq k,l}(w_i*(1-\mu_i))^p + ((w_k-\epsilon)*(1-\mu_k))^p + ((w_l+\epsilon)*(1-\mu_l))^p}{\sum_{i\neq k,l} w_i^p + (w_k-\epsilon)^p + (w_l+\epsilon)^p}\right)^{\frac{1}{p}}$$

Substitution: $c_1 := \sum_{i\neq k,l}(w_i * (1-\mu_i))^p$
$c_2 := \sum_{i\neq k,l} w_i^p$

$$\Leftrightarrow \quad \frac{c_1 + (w_k*(1-\mu_k))^p + (w_l*(1-\mu_l))^p}{c_2 + w_k^p + w_l^p}$$
$$\geq \frac{c_1 + ((w_k-\epsilon)*(1-\mu_k))^p + ((w_l+\epsilon)*(1-\mu_l))^p}{c_2 + (w_k-\epsilon)^p + (w_l+\epsilon)^p}$$

$$\Leftrightarrow \quad (c_1 + w_k^p(1-\mu_k)^p + w_l^p(1-\mu_l)^p) * (c_2 + (w_k-\epsilon)^p + (w_l+\epsilon)^p)$$
$$\geq (c_1 + (w_k-\epsilon)^p(1-\mu_k)^p + (w_l+\epsilon)^p(1-\mu_l)^p) * (c_2 + w_k^p + w_l^p)$$

$$\Leftrightarrow \quad c_1c_2 + c_1(w_k-\epsilon)^p + c_1(w_l+\epsilon)^p$$
$$+w_k^p(1-\mu_k)^p c_2 + w_k^p(1-\mu_k)^p(w_k-\epsilon)^p + w_k^p(1-\mu_k)^p(w_l+\epsilon)^p$$
$$+w_l^p(1-\mu_l)^p c_2 + w_l^p(1-\mu_l)^p(w_k-\epsilon)^p + w_l^p(1-\mu_l)^p(w_l+\epsilon)^p$$
$$\geq c_1c_2 + c_1w_k^p + c_1w_l^p$$
$$+(w_k-\epsilon)^p(1-\mu_k)^p c_2 + (w_k-\epsilon)^p(1-\mu_k)^p w_k^p + (w_k-\epsilon)^p(1-\mu_k)^p w_l^p$$
$$+(w_l+\epsilon)^p(1-\mu_l)^p c_2 + (w_l+\epsilon)^p(1-\mu_l)^p w_k^p + (w_l+\epsilon)^p(1-\mu_l)^p w_l^p$$

$$\Leftrightarrow \quad c_1(w_k-\epsilon)^p + c_1(w_l+\epsilon)^p$$
$$+w_k^p(1-\mu_k)^p c_2 + w_k^p(1-\mu_k)^p(w_l+\epsilon)^p$$
$$+w_l^p(1-\mu_l)^p c_2 + w_l^p(1-\mu_l)^p(w_k-\epsilon)^p$$
$$\geq c_1w_k^p + c_1w_l^p$$
$$+(w_k-\epsilon)^p(1-\mu_k)^p c_2 + (w_k-\epsilon)^p(1-\mu_k)^p w_l^p$$
$$+(w_l+\epsilon)^p(1-\mu_l)^p c_2 + (w_l+\epsilon)^p(1-\mu_l)^p w_k^p$$

Wie nicht unschwer zu erkennen, kann diese Ungleichung nicht weiter vereinfacht werden, und insbesondere nicht in die in Def. 3.13 geforderte Voraussetzung umgeformt werden.

q.e.d

B.6 Assoziativität des WAO-Operators

Der WAO-Operator ist nicht assoziativ. Genauer: Es lässt sich das zusätzlich notwendige Gewicht $w_{a,b}$ nicht unabhängig von den anderen Gewichten und den Zugehörigkeitsgraden bestimmen.

Beweis:

$$\begin{aligned}
z.z.: \quad & WAO_\lambda(\langle w_{a,b}, WAO_\lambda(\langle w_a,\mu_a\rangle,\langle w_b,\mu_b\rangle)\rangle,\langle w_c,\mu_c\rangle) \\
& = WAO_\lambda(\langle w_a,\mu_a\rangle,\langle w_b,\mu_b\rangle,\langle w_c,\mu_c\rangle)
\end{aligned}$$

$$\Leftrightarrow \quad \left(\frac{\left(w_{a,b} * \left(\frac{(w_a*\mu_a)^\lambda+(w_b*\mu_b)^\lambda}{w_a^\lambda+w_b^\lambda}\right)^{\frac{1}{\lambda}}\right)^\lambda + (w_c*\mu_c)^\lambda}{w_{a,b}^\lambda + w_c^\lambda}\right)^{\frac{1}{\lambda}} = \left(\frac{(w_a*\mu_a)^\lambda + (w_b*\mu_b)^\lambda + (w_c*\mu_c)^\lambda}{w_a^\lambda + w_b^\lambda + w_c^\lambda}\right)^{\frac{1}{\lambda}}$$

$$\Leftrightarrow \quad \frac{w_{a,b}^\lambda * \frac{(w_a*\mu_a)^\lambda+(w_b*\mu_b)^\lambda}{w_a^\lambda+w_b^\lambda} + (w_c*\mu_c)^\lambda}{w_{a,b}^\lambda + w_c^\lambda} = \frac{(w_a*\mu_a)^\lambda + (w_b*\mu_b)^\lambda + (w_c*\mu_c)^\lambda}{w_a^\lambda + w_b^\lambda + w_c^\lambda}$$

$$\begin{aligned}
\Leftrightarrow \quad & (\frac{w_{a,b}^\lambda * (w_a*\mu_a)^\lambda + w_{a,b}^\lambda * (w_b*\mu_b)^\lambda}{w_a^\lambda + w_b^\lambda} + (w_c*\mu_c)^\lambda) * (w_a^\lambda + w_b^\lambda + w_c^\lambda) \\
& = ((w_a*\mu_a)^\lambda + (w_b*\mu_b)^\lambda + (w_c*\mu_c)^\lambda) * (w_{a,b}^\lambda + w_c^\lambda)
\end{aligned}$$

$$\begin{aligned}
\Leftrightarrow \quad & (w_{a,b}^\lambda * (w_a*\mu_a)^\lambda + w_{a,b}^\lambda * (w_b*\mu_b)^\lambda + (w_c*\mu_c)^\lambda * (w_a^\lambda + w_b^\lambda)) * (w_a^\lambda + w_b^\lambda + w_c^\lambda) \\
& = ((w_a*\mu_a)^\lambda + (w_b*\mu_b)^\lambda + (w_c*\mu_c)^\lambda) * (w_{a,b}^\lambda + w_c^\lambda) * (w_a^\lambda + w_b^\lambda)
\end{aligned}$$

$$\begin{aligned}
\Leftrightarrow \quad & (w_{a,b}^\lambda w_a^\lambda \mu_a^\lambda + w_{a,b}^\lambda w_b^\lambda \mu_b^\lambda + w_c^\lambda \mu_c^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_b^\lambda) * (w_a^\lambda + w_b^\lambda + w_c^\lambda) \\
& = (w_a^\lambda \mu_a^\lambda + w_b^\lambda \mu_b^\lambda + w_c^\lambda \mu_c^\lambda) * (w_{a,b}^\lambda w_a^\lambda + w_{a,b}^\lambda w_b^\lambda + w_c^\lambda w_a^\lambda + w_c^\lambda w_b^\lambda)
\end{aligned}$$

$$\begin{aligned}
\Leftrightarrow \quad & w_{a,b}^\lambda w_a^\lambda \mu_a^\lambda w_a^\lambda + w_{a,b}^\lambda w_a^\lambda \mu_a^\lambda w_b^\lambda + w_{a,b}^\lambda w_a^\lambda \mu_a^\lambda w_c^\lambda + w_{a,b}^\lambda w_b^\lambda \mu_b^\lambda w_a^\lambda + w_{a,b}^\lambda w_b^\lambda \mu_b^\lambda w_b^\lambda + w_{a,b}^\lambda w_b^\lambda \mu_b^\lambda w_c^\lambda + \\
& \quad w_c^\lambda \mu_c^\lambda w_a^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_a^\lambda w_b^\lambda + w_c^\lambda \mu_c^\lambda w_a^\lambda w_c^\lambda + w_c^\lambda \mu_c^\lambda w_b^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_b^\lambda w_b^\lambda + w_c^\lambda \mu_c^\lambda w_b^\lambda w_c^\lambda \\
& = w_a^\lambda \mu_a^\lambda w_{a,b}^\lambda w_a^\lambda + w_a^\lambda \mu_a^\lambda w_{a,b}^\lambda w_b^\lambda + w_a^\lambda \mu_a^\lambda w_c^\lambda w_a^\lambda + w_a^\lambda \mu_a^\lambda w_c^\lambda w_b^\lambda + \\
& \quad w_b^\lambda \mu_b^\lambda w_{a,b}^\lambda w_a^\lambda + w_b^\lambda \mu_b^\lambda w_{a,b}^\lambda w_b^\lambda + w_b^\lambda \mu_b^\lambda w_c^\lambda w_a^\lambda + w_b^\lambda \mu_b^\lambda w_c^\lambda w_b^\lambda + \\
& \quad w_c^\lambda \mu_c^\lambda w_{a,b}^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_{a,b}^\lambda w_b^\lambda + w_c^\lambda \mu_c^\lambda w_c^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_c^\lambda w_b^\lambda
\end{aligned}$$

$$\begin{aligned}
\Leftrightarrow \quad & w_{a,b}^\lambda w_a^\lambda \mu_a^\lambda w_c^\lambda + w_{a,b}^\lambda w_b^\lambda \mu_b^\lambda w_c^\lambda + w_c^\lambda \mu_c^\lambda w_a^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_a^\lambda w_b^\lambda + w_c^\lambda \mu_c^\lambda w_b^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_b^\lambda w_b^\lambda \\
& = w_a^\lambda \mu_a^\lambda w_c^\lambda w_a^\lambda + w_a^\lambda \mu_a^\lambda w_c^\lambda w_b^\lambda + w_b^\lambda \mu_b^\lambda w_c^\lambda w_a^\lambda + w_b^\lambda \mu_b^\lambda w_c^\lambda w_b^\lambda + w_c^\lambda \mu_c^\lambda w_{a,b}^\lambda w_a^\lambda + w_c^\lambda \mu_c^\lambda w_{a,b}^\lambda w_b^\lambda
\end{aligned}$$

$$\Leftrightarrow \quad w_{a,b}^{\lambda} w_a^{\lambda} \mu_a^{\lambda} + w_{a,b}^{\lambda} w_b^{\lambda} \mu_b^{\lambda} + \mu_c^{\lambda} w_a^{\lambda} w_a^{\lambda} + \mu_c^{\lambda} w_a^{\lambda} w_b^{\lambda} + \mu_c^{\lambda} w_b^{\lambda} w_a^{\lambda} + \mu_c^{\lambda} w_b^{\lambda} w_b^{\lambda}$$
$$= w_a^{\lambda} \mu_a^{\lambda} w_a^{\lambda} + w_a^{\lambda} \mu_a^{\lambda} w_b^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} w_a^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} w_b^{\lambda} + \mu_c^{\lambda} w_{a,b}^{\lambda} w_a^{\lambda} + \mu_c^{\lambda} w_{a,b}^{\lambda} w_b^{\lambda}$$

$$\Leftrightarrow \quad w_{a,b}^{\lambda} w_a^{\lambda} \mu_a^{\lambda} + w_{a,b}^{\lambda} w_b^{\lambda} \mu_b^{\lambda} - \mu_c^{\lambda} w_{a,b}^{\lambda} w_a^{\lambda} - \mu_c^{\lambda} w_{a,b}^{\lambda} w_b^{\lambda}$$
$$= w_a^{\lambda} \mu_a^{\lambda} w_a^{\lambda} + w_a^{\lambda} \mu_a^{\lambda} w_b^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} w_a^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} w_b^{\lambda} - \mu_c^{\lambda} w_a^{\lambda} w_a^{\lambda} - \mu_c^{\lambda} w_a^{\lambda} w_b^{\lambda} - \mu_c^{\lambda} w_b^{\lambda} w_a^{\lambda} - \mu_c^{\lambda} w_b^{\lambda} w_b^{\lambda}$$

$$\Leftrightarrow \quad w_{a,b}^{\lambda} = \frac{w_a^{\lambda} \mu_a^{\lambda} w_a^{\lambda} + w_a^{\lambda} \mu_a^{\lambda} w_b^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} w_a^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} w_b^{\lambda} - \mu_c^{\lambda} w_a^{\lambda} w_a^{\lambda} - \mu_c^{\lambda} w_a^{\lambda} w_b^{\lambda} - \mu_c^{\lambda} w_b^{\lambda} w_a^{\lambda} - \mu_c^{\lambda} w_b^{\lambda} w_b^{\lambda}}{w_a^{\lambda} \mu_a^{\lambda} + w_b^{\lambda} \mu_b^{\lambda} - \mu_c^{\lambda} w_a^{\lambda} - \mu_c^{\lambda} w_b^{\lambda}}$$

$w_{a,b}$ kann also nicht unabhängig von den anderen Parametern bestimmt werden und ist insbesondere sogar abhängig von den Zugehörigkeitsgraden μ_a und μ_b!

B.7 Grenzbedingung des WAO-Operators

Der WAO-Operator erfüllt die Grenzbedingung nach Def. 3.15.8.

Beweis:

$$WAO_\lambda(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) = \left(\frac{\sum (w_i * \mu_i)^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} \leq \left(\frac{\sum (w_i * \max_j\{\mu_j\})^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} = \left(\frac{(\max_j\{\mu_j\})^\lambda * \sum w_i^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} = \max_j\{\mu_j\}$$

$$WAO_\lambda(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) = \left(\frac{\sum (w_i * \mu_i)^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} \geq \left(\frac{\sum (w_i * \min_j\{\mu_j\})^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} = \left(\frac{(\min_j\{\mu_j\})^\lambda * \sum w_i^\lambda}{\sum w_i^\lambda}\right)^{\frac{1}{\lambda}} = \min_j\{\mu_j\}$$

B.8 Strenge ∃-Monotonie des WAO-Operators

Der WAO-Operator erfüllt die strenge ∃-Monotonie bezüglich der Gewichte.

Beweis:

z.z.: $WAO_\lambda(\langle w_1, \mu_1\rangle, ..., \langle w_n, \mu_n\rangle) < WAO_\lambda(\langle w_1, \mu_1\rangle, ..., \langle w'_n, \mu_n\rangle)$
für $w_n < w'_n$ und $\mu_n > WAO_\lambda(\langle w_1, \mu_1\rangle, ..., \langle w_{n-1}, \mu_{n-1}\rangle)$

$$
\begin{aligned}
& WAO_\lambda(\langle w_1,\mu_1\rangle,...,\langle w_n,\mu_n\rangle) < WAO_\lambda(\langle w_1,\mu_1\rangle,...,\langle w'_n,\mu_n\rangle) \\
\Leftrightarrow\quad & \left(\frac{c_1+(w_n\mu_n)^\lambda}{c_2+w_n^\lambda}\right)^{\frac{1}{\lambda}} < \left(\frac{c_1+(w'_n\mu_n)^\lambda}{c_2+w'^\lambda_n}\right)^{\frac{1}{\lambda}} \quad mit\ c_1=\sum_{i=1}^{n-1}(w_i\mu_i)^\lambda\ und\ c_2=\sum_{i=1}^{n-1}w_i^\lambda \\
\Leftrightarrow\quad & (c_1+(w_n\mu_n)^\lambda)(c_2+w'^\lambda_n) < (c_1+(w'_n\mu_n))^\lambda(c_2+w_n^\lambda) \\
\Leftrightarrow\quad & c_1c_2+c_1w'^\lambda_n+w_n^\lambda\mu_n^\lambda c_2+w_n^\lambda\mu_n^\lambda w'^\lambda_n < c_1c_2+c_1w_n^\lambda+w'^\lambda_n\mu_n^\lambda c_2+w'^\lambda_n\mu_n^\lambda w_n^\lambda \\
\Leftrightarrow\quad & c_1w'^\lambda_n+w_n^\lambda\mu_n^\lambda c_2 < c_1w_n^\lambda+w'^\lambda_n\mu_n^\lambda c_2 \\
\Leftrightarrow\quad & c_1w'^\lambda_n-w'^\lambda_n\mu_n^\lambda c_2 < c_1w_n^\lambda-w_n^\lambda\mu_n^\lambda c_2 \\
\Leftrightarrow\quad & w'^\lambda_n*(c_1-\mu_n^\lambda c_2) < w_n^\lambda*(c_1-\mu_n^\lambda c_2) \\
\Leftrightarrow\quad & c_1-\mu_n^\lambda c_2<0 \quad\wedge\quad w_n<w'_n \\
\Leftrightarrow\quad & \mu_n^\lambda > \frac{c_1}{c_2} \quad\wedge\quad w_n<w'_n \\
\Leftrightarrow\quad & \mu_n > \left(\frac{\sum_{i=1}^{n-1}(w_i\mu_i)^\lambda}{\sum_{i=1}^{n-1}w_i^\lambda}\right)^{\frac{1}{\lambda}} \quad\wedge\quad w_n<w'_n \\
\Leftrightarrow\quad & \mu_n > WAO_\lambda(\langle w_1,\mu_1\rangle,...,\langle w_{n-1},\mu_{n-1}\rangle) \quad\wedge\quad w_n<w'_n
\end{aligned}
$$

Analog: $WAO_\lambda(\langle w_1,\mu_1\rangle,...,\langle w_n,\mu_n\rangle) > WAO_\lambda(\langle w_1,\mu_1\rangle,...,\langle w'_n,\mu_n\rangle)$
für $w_n < w'_n$ und $\mu_n < WAO_\lambda(\langle w_1,\mu_1\rangle,...,\langle w_{n-1},\mu_{n-1}\rangle)$

B.9 Strenge K-Monotonie des WAO-Operators

Der WAO-Operator erfüllt die strenge K-Monotonie.

Beweis:

z.z.: $WAO_\lambda(\langle w_k,\mu_k\rangle,\langle w_b,\mu_l\rangle,...) < WAO_\lambda(\langle w_k-\epsilon,\mu_k\rangle,\langle w_l+\epsilon,\mu_l\rangle,...)$
für $\mu_k<\mu_l$ und $\forall\epsilon>0$

$$
\begin{aligned}
& WAO_\lambda(...,\langle w_k,\mu_k\rangle,\langle w_l,\mu_l\rangle,...) < WAO_\lambda(...,\langle w_k-\epsilon,\mu_k\rangle,\langle w_l+\epsilon,\mu_l\rangle,...) \\
\Leftrightarrow\quad & \left(\frac{\sum_i(w_i*\mu_i)^\lambda}{\sum_i w_i^\lambda}\right)^{\frac{1}{\lambda}} < \left(\frac{\sum_{i\neq k,l}(w_i*\mu_i)^\lambda+(w_k-\epsilon)^\lambda*\mu_k^\lambda+(w_l+\epsilon)^\lambda*\mu_l^\lambda}{\sum_{i\neq k,l}w_i^\lambda+(w_k-\epsilon)^\lambda+(w_l+\epsilon)^\lambda}\right)^{\frac{1}{\lambda}} \\
\Leftrightarrow\quad & \frac{w_k^\lambda*\mu_k^\lambda+w_l^\lambda*\mu_l^\lambda}{w_k^\lambda+w_l^\lambda} < \frac{(w_k-\epsilon)^\lambda*\mu_k^\lambda+(w_l+\epsilon)^\lambda*\mu_l^\lambda}{(w_k-\epsilon)^\lambda+(w_l+\epsilon)^\lambda} \\
\Leftrightarrow\quad & (w_k^\lambda*\mu_k^\lambda+w_l^\lambda*\mu_l^\lambda)((w_k-\epsilon)^\lambda+(w_l+\epsilon)^\lambda)
\end{aligned}
$$

$$
\begin{aligned}
& < ((w_k - \epsilon)^\lambda * \mu_k^\lambda + (w_l + \epsilon)^\lambda * \mu_l^\lambda)(w_k^\lambda + w_l^\lambda) \\
\Leftrightarrow \quad & w_k^\lambda \mu_k^\lambda (w_k - \epsilon)^\lambda + w_k^\lambda \mu_k^\lambda (w_l + \epsilon)^\lambda + w_l^\lambda \mu_l^\lambda (w_k - \epsilon)^\lambda + w_l^\lambda \mu_l^\lambda (w_l + \epsilon)^\lambda \\
& < (w_k - \epsilon)^\lambda \mu_k^\lambda w_k^\lambda + (w_l + \epsilon)^\lambda \mu_l^\lambda w_k^\lambda + (w_k - \epsilon)^\lambda \mu_k^\lambda w_l^\lambda + (w_l + \epsilon)^\lambda \mu_l^\lambda w_l^\lambda \\
\Leftrightarrow \quad & w_k^\lambda \mu_k^\lambda (w_l + \epsilon)^\lambda + w_l^\lambda \mu_l^\lambda (w_k - \epsilon)^\lambda < (w_l + \epsilon)^\lambda \mu_l^\lambda w_k^\lambda + (w_k - \epsilon)^\lambda \mu_k^\lambda w_l^\lambda \\
\Leftrightarrow \quad & w_k^\lambda \mu_k^\lambda (w_l + \epsilon)^\lambda - (w_l + \epsilon)^\lambda \mu_l^\lambda w_k^\lambda < (w_k - \epsilon)^\lambda \mu_k^\lambda w_l^\lambda - w_l^\lambda \mu_l^\lambda (w_k - \epsilon)^\lambda \\
\Leftrightarrow \quad & (\mu_k^\lambda - \mu_l^\lambda) w_k^\lambda (w_l + \epsilon)^\lambda < (\mu_k^\lambda - \mu_l^\lambda) w_l^\lambda (w_k - \epsilon)^\lambda \\
\Leftrightarrow \quad & w_k (w_l + \epsilon) > w_l (w_k - \epsilon) \\
\Leftrightarrow \quad & w_k w_l + w_k \epsilon > w_l w_k - w_l \epsilon \\
\Leftrightarrow \quad & +w_k > -w_l
\end{aligned}
$$

B.10 Parametrisierung des WAO-Operators

Der WAO-Operator erfüllt die Parametrisierungs-Eigenschaft.

Beweis:

z.z.: $\exists \lambda_1, \lambda_2 \forall w$:

$$
WAO_{\lambda_1}(\langle w, 1\rangle, \underbrace{\langle \frac{w}{n}, 0\rangle, ..., \langle \frac{w}{n}, 0\rangle}_{n-mal}) < WAO_{\lambda_1}(\langle w, 0\rangle, \underbrace{\langle \frac{w}{n}, 1\rangle, ..., \langle \frac{w}{n}, 1\rangle}_{n-mal})
$$

$\wedge$

$$
WAO_{\lambda_2}(\langle w, 1\rangle, \underbrace{\langle \frac{w}{n}, 0\rangle, ..., \langle \frac{w}{n}, 0\rangle}_{n-mal}) > WAO_{\lambda_2}(\langle w, 0\rangle, \underbrace{\langle \frac{w}{n}, 1\rangle, ..., \langle \frac{w}{n}, 1\rangle}_{n-mal})
$$

für $n \geq 2$

$$
\begin{aligned}
& WAO_{\lambda_1}(\langle w, 1\rangle, \underbrace{\langle \frac{w}{n}, 0\rangle, ..., \langle \frac{w}{n}, 0\rangle}_{n-mal}) < WAO_{\lambda_1}(\langle w, 0\rangle, \underbrace{\langle \frac{w}{n}, 1\rangle, ..., \langle \frac{w}{n}, 1\rangle}_{n-mal}) \\
\Leftrightarrow \quad & \left(\frac{(w * 1)^{\lambda_1} + \sum_{i=1}^{n} (\frac{w}{n} * 0)^{\lambda_1}}{w^{\lambda_1} + \sum_{i=1}^{n} (\frac{w}{n})^{\lambda_1}} \right)^{\frac{1}{\lambda_1}} < \left(\frac{(w * 0)^{\lambda_1} + \sum_{i=1}^{n} ((\frac{w}{n}) * 1)^{\lambda_1}}{w^{\lambda_1} + \sum_{i=1}^{n} (\frac{w}{n})^{\lambda_1}} \right)^{\frac{1}{\lambda_1}} \\
\Leftrightarrow \quad & w^{\lambda_1} < n * ((\frac{w}{n}) * 1)^{\lambda_1} \\
\Leftrightarrow \quad & w^{\lambda_1} < n \frac{w^{\lambda_1}}{n^{\lambda_1}} \\
\Leftrightarrow \quad & 1 < n \frac{1}{n^{\lambda_1}}
\end{aligned}
$$

$$\Leftrightarrow \quad 1 < n^{1-\lambda_1}$$
$$\Leftrightarrow \quad 0 < 1 - \lambda_1$$
$$\Leftrightarrow \quad \lambda_1 < 1$$

Analog ergibt sich $\lambda_2 > 1$

B.11 Integrale über die Normalverteilung

Anstelle der Normalverteilung

$$g(x,y) = F_{norm} * e^{-\frac{1}{2}*(\binom{x}{y}-\vec{\mu})^T K^{-1}(\binom{x}{y}-\vec{\mu})}$$
$$\text{mit } \vec{\mu} = \binom{0.5}{0.5},\ K = \begin{pmatrix} 1 & k \\ k & 1 \end{pmatrix},\ \text{z.B. } k = -0.7$$

wird die Approximation

$$g(x,y) \approx F_{norm} * \left(\frac{1}{2} - \frac{1}{2\pi}\left(\left(\binom{x}{y} - \vec{\mu}\right)^T K^{-1}\left(\binom{x}{y} - \vec{\mu}\right)\right) + Restglieder\right)$$

mit

$$F_{norm} = \frac{1}{\int_0^1 \int_0^1 \frac{1}{2} - \frac{1}{2\pi}\left(\left(\binom{x}{y} - \vec{\mu}\right)^T K^{-1}\left(\binom{x}{y} - \vec{\mu}\right)\right) dx\, dy} = \frac{2(k^2-1)\pi}{\frac{1}{6} - \pi + \pi k^2}$$

verwendet. Abbildung B.1 zeigt die Normalverteilung und ihre für die Praxis sehr gute Approximation.

Die Integrale $G_x(a)$, $G_y(a)$, $G(a)$, $A(a)$, $M(a)$ und deren Inverse berechnen sich damit wie folgt:

$$\begin{aligned}
G_x(a) &= \int_0^1 \int_a^1 g(x,y)\, dx\, dy \\
&= \int_0^1 \int_a^1 F_{norm} * \left(\frac{1}{2} - \frac{1}{2\pi}\left(\left(\binom{x}{y} - \vec{\mu}\right)^T K^{-1}\left(\binom{x}{y} - \vec{\mu}\right)\right)\right) dx\, dy \\
&= F_{norm} \int_0^1 \int_a^1 \frac{1}{4\pi * (-1+k^2)} * (-2\pi + 2\pi k^2 + 2x^2 - 2x + 1 \\
&\quad -4kxy + 2ky + 2kx - k + 2y^2 - 2y)\, dx\, dy \\
&= -\frac{2(k^2-1)\pi}{\frac{1}{6} - \pi + \pi k^2} * \frac{6\pi - 6\pi k^2 - 1 - 6a\pi + 6a\pi k^2 + 2a^3 + 2a - 3a^2}{12\pi * (-1+k^2)} \\
&= -\frac{6\pi - 6\pi k^2 - 1 - 6a\pi + 6a\pi k^2 + 2a^3 + 2a - 3a^2}{1 - 6\pi + 6\pi k^2} \\
&\quad \text{für } k = -0.7 \text{ (vgl. Abb. B.2):} \\
&\approx 1 - a
\end{aligned}$$

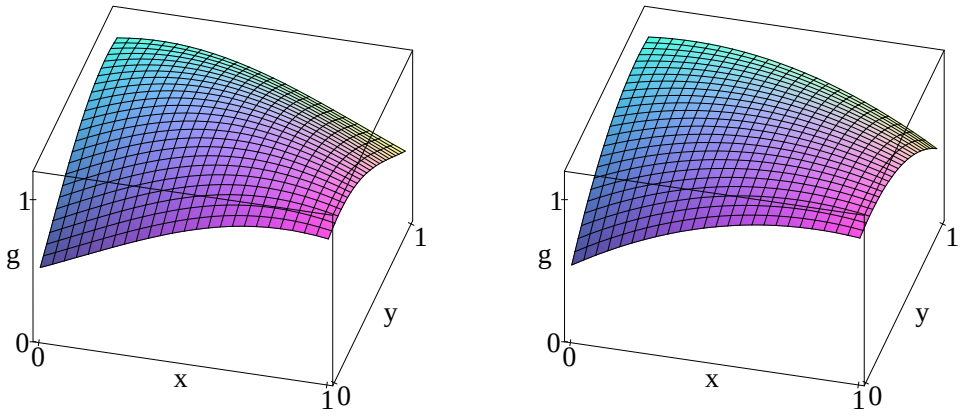


Abbildung B.1: Negativ korrelierte, normalverteilte Dichtefunktion (links) und Approximation (rechts)

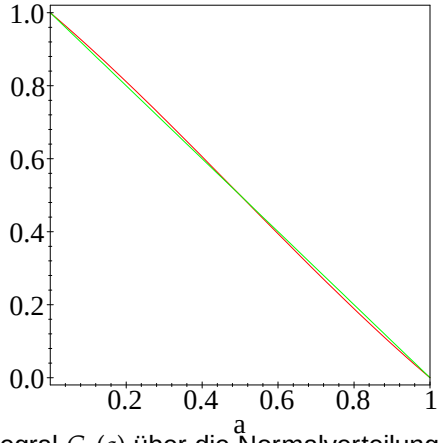


Abbildung B.2: Integral $G_x(a)$ über die Normalverteilung (rot) und Approximation (grün).

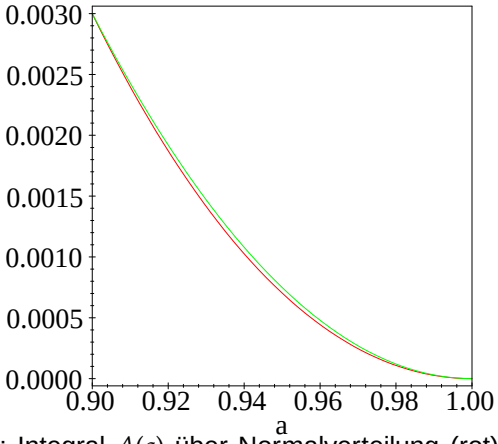


Abbildung B.3: Integral $A(a)$ über Normalverteilung (rot) und Approximation (grün).

Analog ergibt sich $G_y(a) \approx 1 - a$ und damit $G(a) \approx 2 - 2a$.

$$
\begin{aligned}
A(a) &= \int_a^1 \int_{a+1-x}^1 g(x,y)\, dy\, dx \\
&= \int_a^1 \int_{a+1-x}^1 F_{norm} * \left(\frac{1}{2} - \frac{1}{2\pi} \left(\left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right)^T K^{-1} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right) \right) \right) dx\, dy \\
&= F_{norm} \int_a^1 \int_{a+1-x}^1 \frac{1}{4\pi * (-1+k^2)} * (-2\pi + 2\pi k^2 + 2x^2 - 2x + 1 \\
&\quad -4kxy + 2ky + 2kx - k + 2y^2 - 2y)\, dx\, dy \\
&= -\frac{1}{2 * (6\pi k^2 - 6\pi + 1)} * (-6\pi k^2 + 6\pi - 1 - 12a\pi + 2a + 12a\pi k^2 + 4a^3 + 2ka \\
&\quad -3a^2 - 6a^2\pi k^2 + ka^4 + 6a^2\pi - 3ka^2 - 2a^4) \\
&\quad \text{für } k = -0.7 \text{ und } a \approx 1 \text{ (vgl. Abb. B.3):} \\
&\approx 0.3 * (1-a)^2
\end{aligned}
$$

Damit erhält man $A^{-1}(A) \approx 1 - \sqrt{\frac{1}{0.3} A}$.

$$
M(a) = \int_a^1 \int_a^1 g(x,y)\, dy\, dx
$$

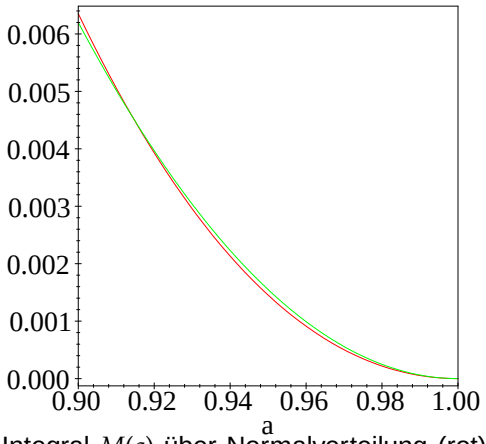


Abbildung B.4: Integral $M(a)$ über Normalverteilung (rot) und Approximation (grün).

$$
\begin{aligned}
&= \int_a^1 \int_a^1 F_{norm} * \left(\frac{1}{2} - \frac{1}{2\pi} \left(\left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right)^T K^{-1} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{\mu} \right) \right) \right) dx\, dy \\
&= F_{norm} \int_a^1 \int_a^1 \frac{1}{4\pi * (-1+k^2)} * (-2\pi + 2\pi k^2 + 2x^2 - 2x + 1 \\
&\quad -4kxy + 2ky + 2kx - k + 2y^2 - 2y)\, dx\, dy \\
&= -\frac{1}{6\pi k^2 - 6\pi + 1} * (-6\pi k^2 + 6\pi - 1 + 12a\pi k^2 - 12a\pi + 10a^3 + 4a - 9a^2 - \\
&\quad 6a^2\pi k^2 + 3ka^4 + 6a^2\pi - 6ka^3 - 4a^4 + 3ka^2) \\
&\quad \text{für } k = -0.7 \text{ und } a \approx 1 \text{ (vgl. Abb. B.3):} \\
&\approx 0.62 * (1-a)^2
\end{aligned}
$$

Damit ergibt sich $M^{-1}(M) \approx 1 - \sqrt{\frac{1}{0.62} M}$.

B.12 Integrale über die Exponentialverteilung

Anstelle der Exponentialverteilung

$$
\begin{aligned}
g(x,y) &= F_{norm} * p^x \\
\text{mit} & \\
F_{norm} &= \frac{ln(p)}{p-1}, \text{ z.B. } p = 0.01
\end{aligned}
$$

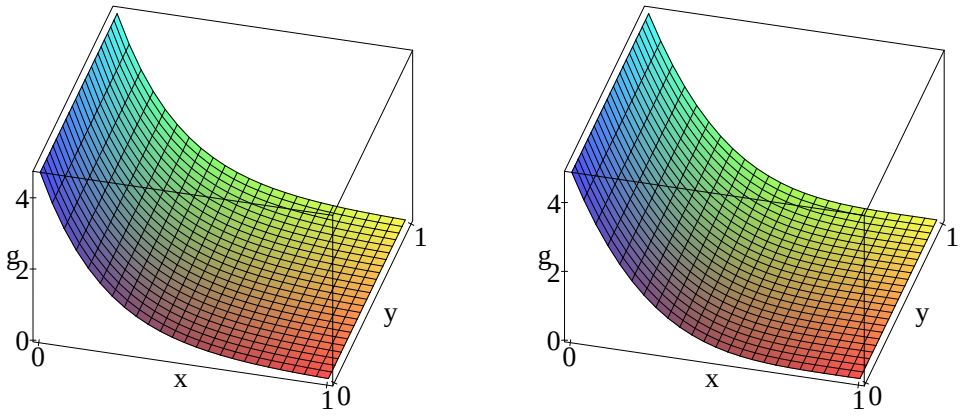


Abbildung B.5: Exponentialverteilte Dichtefunktion (links) und Approximation (rechts)

wir die Approximation

$$g(x,y) \approx F_{norm} * \left(p + (1-x)^{-2*log_{10}(p)} + Restglieder\right) = F_{norm} * (0.01 + (1-x)^4)$$

mit

$$F_{norm} = \frac{1}{\int_0^1 \int_0^1 p + (1-x)^{-2*log_{10}(p)}\, dx\, dy} = \frac{1}{p - \frac{1}{2log_{10}(p)-1}} = \frac{1}{0.21}$$

verwendet. Abbildung B.5 zeigt die Exponentialverteilung und ihre für die Praxis sehr gute Approximation.

Die Integrale $G(a)$, $A(a)$, $M(a)$ und deren Inverse berechnen sich damit wie folgt:

$$\begin{aligned}
G(a) &= G_x(a) + G_y(a) \\
&= \int_0^1 \int_a^1 g(x,y)\, dx\, dy + \int_0^1 \int_a^1 g(x,y)\, dy\, dx \\
&= \int_0^1 \int_a^1 \frac{1}{0.21} * (0.01 + (1-x)^4)\, dx\, dy + \int_0^1 \int_a^1 \frac{1}{0.21} * (0.01 + (1-x)^4)\, dy\, dx \\
&= \frac{22}{21}(1-a) + \frac{20}{21}(1-a)^5 \\
&\quad \text{für } a \approx 1\text{:} \\
&\approx 1 - a
\end{aligned}$$

$$
\begin{aligned}
A(a) &= \int_a^1 \int_{a+1-x}^1 g(x,y)\,dy\,dx \\
&= \int_a^1 \int_{a+1-x}^1 \frac{1}{0.21} * (0.01 + (1-x)^4)\,dx\,dy \\
&= \frac{1}{42}(1-a)^2 + \frac{1}{6.3}(1-a)^6 \\
&\quad \text{für } a \approx 1\text{:} \\
&\approx \frac{1}{42}(1-a)^2
\end{aligned}
$$

Damit ergibt sich $A^{-1}(A) = 1 - \sqrt{42A}$.

$$
\begin{aligned}
M(a) &= \int_a^1 \int_a^1 g(x,y)\,dy\,dx \\
&= \int_a^1 \int_a^1 \frac{1}{0.21} * (0.01 + (1-x)^4)\,dx\,dy \\
&= \frac{1}{21}(1-a)^2 + \frac{1}{1.05}(1-a)^6 \\
&\quad \text{für } a \approx 1\text{:} \\
&= \frac{1}{21}(1-a)^2
\end{aligned}
$$

Damit ergibt sich $M^{-1} = 1 - \sqrt{21M}$.

C

Graphen des WAO-Operators

Abbildungen C.1-C.3 zeigen Graphen des WAO-Operators für $n = 2$ in Abhängigkeit von $\langle w_1, \mu_1 \rangle$ für $w_2 = 1$ und $\mu_2 = 1$ bzw. 0 und in Abhängigkeit von w_1 und w_2, jeweils für verschiedene Werte von λ.

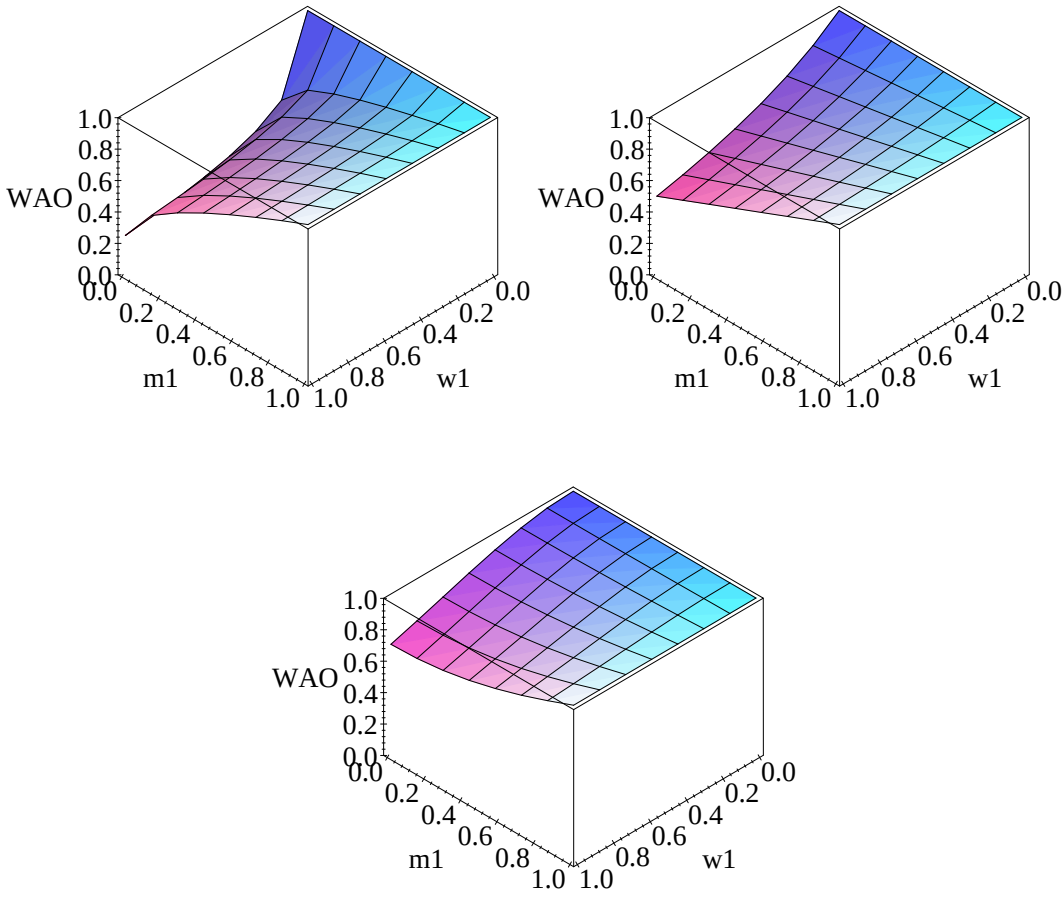


Abbildung C.1: WAO_λ-Operator in Abhängigkeit von $\langle w_1, \mu_1 \rangle$ für $w_2 = 1$ und $\mu_2 = 1$ für $\lambda = 0.5$, $\lambda = 1$ und $\lambda = 2$.

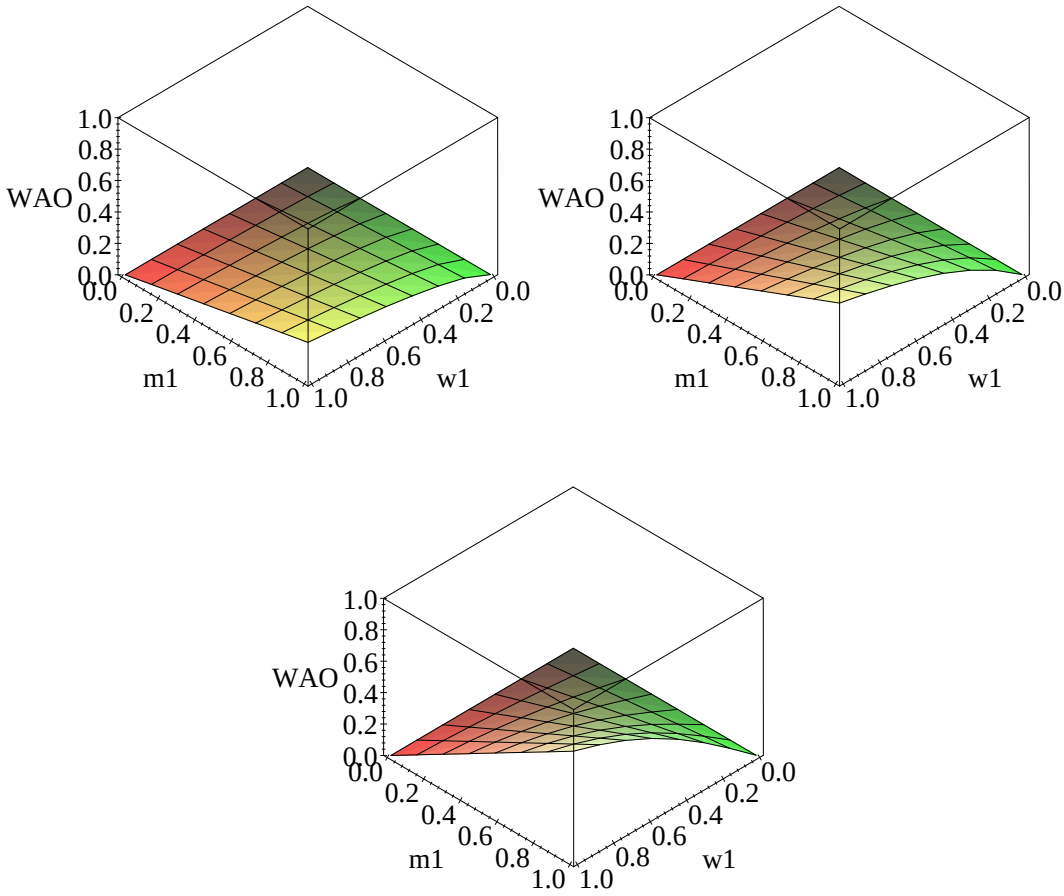


Abbildung C.2: WAO_λ-Operator in Abhängigkeit von $\langle w_1, \mu_1 \rangle$ für $w_2 = 1$ und $\mu_2 = 0$ für $\lambda = 0.5$, $\lambda = 1$ und $\lambda = 2$.

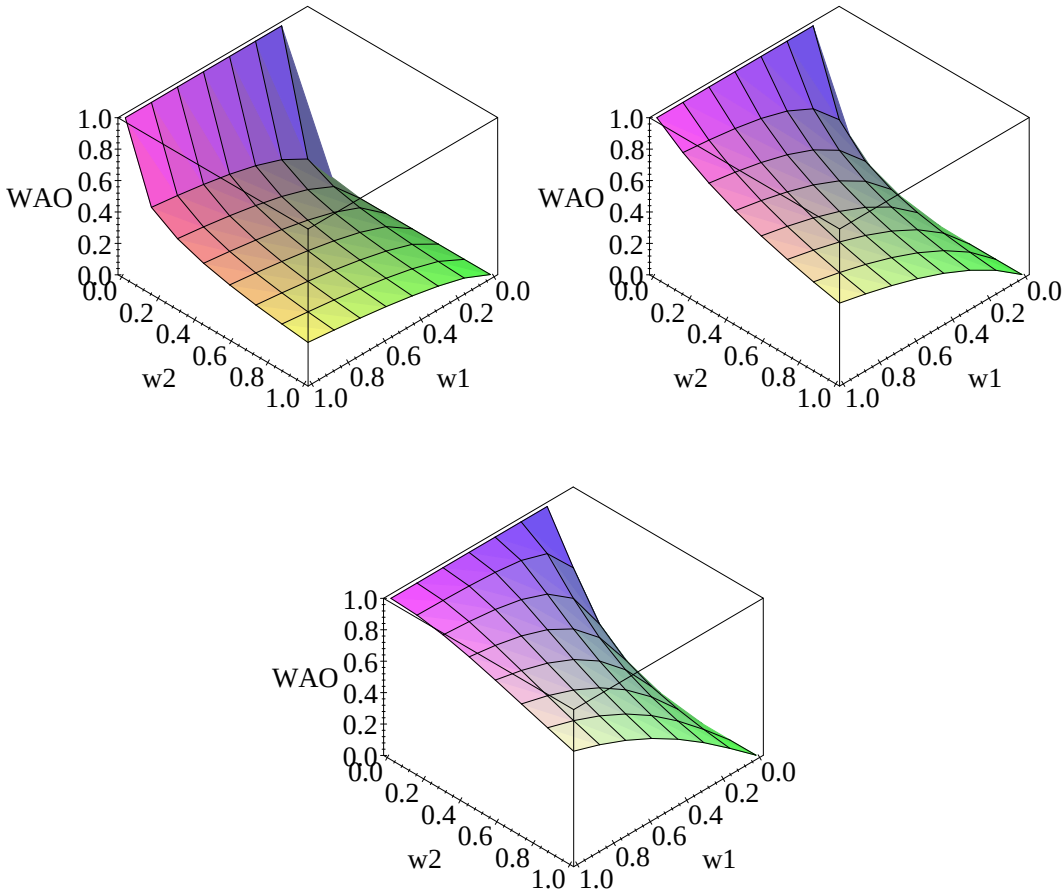


Abbildung C.3: WAO_λ-Operator in Abhängigkeit von w_1 und w_2 für $\mu_1 = 1$, $\mu_2 = 0$ und $\lambda = 0.5$, $\lambda = 1$ und $\lambda = 2$.

Literaturverzeichnis

[1] A. Ait-Bouziad and H. Kassel. An Improved Algorithm for Retrieving Fuzzy Information from Two Systems. *Information Processing Letters*, 65:63–66, 1998.

[2] J. Srivastava B. Mobasher, R. Cooley. Automatic personalization based on web usage mining. http://maya.cs.depaul.edu/~mobasher/personalization.

[3] J. Barwise and R. Cooper. Generalized Quantifiers And Natural Language. *Linguistics & Philosophy*, 4:159–218, 1981.

[4] G. Binder, M. Stahl, and L. Faulborn. Projektbericht Vergleichsuntersuchung MESSENGER - FULCRUM. http://www.social-science.gesis.de, 2000.

[5] G. Bordogna and G. Pasi. Controlling retrieval through a user-adaptive representation of documents. *International Journal of Approximate Reasoning*, 12:317–339, 1995.

[6] G. Bordogna and G. Pasi. A user-adaptive neural network supporting a rule-based relevance feedback. *Fuzzy sets and systems*, 82:201–211, 1996.

[7] BRS. http://www.opentext.com/dataware/brs_search.html. Dataware Technologies, Inc.

[8] C. Buckley, J. Allan, and G. Salton. Automatic routing and retrieval using smart: TREC-2. *Information Processing & Management*, 31(3):315–326, 1995.

[9] J. P. Callan, W. B. Croft, and J. Broglio. TREC and TIPSTER experiments with INQUERY. *Information Processing & Management*, 31(3):327–343, 1995.

[10] K. S. Candan, W.-S. Li, and M. L. Priya. Similarity-based ranking and query processing in multimedia databases. *Data & Knowledge Engineering*, 35:259–298, 2000.

[11] M. J. Carey, L. M. Haas, and P. M. Schwarz et al. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. *Proceedings of the 5th International Workshop on Research Issues in Data Engineering: Distributed Object Management*, pages 124–131, 1995.

[12] S. C. Cater and D. H. Kraft. A Generalization and Clarification of the Waller-Kraft Wish List. *Information Processing & Management*, 25(1):15–25, 1989.

[13] CGI. http://www.w3.org/CGI/.

[14] S. Chaudhuri and L. Gravano. Optimizing Queries over Multimedia Repositories. *Proceedings of the ACM SIGMOD Conference*, pages 91–102, 1996.

[15] P.-M. Chen and F.-C. Kuo. An information retrieval system based on a user profile. *Systems and Software*, 54:3–8, 2000.

[16] W. F. Cody, L. M. Haas, and W. Niblack et al. Querying Multimedia Data from Multiple Repositories by Content: the Garlic Project. *Proceedings of the 3rd Working Conference on Visual Database Systems*, 1995.

[17] W. S. Cooper. The paradoxical role of unexamined documents in the evaluation of retrieval effectiveness. *Information Processing & Management*, 12:367–375, 1976.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, 2001.

[19] R. Fagin. Fuzzy Queries in Multimedia Database Systems. *Proceedings of the ACM Symposium on Principles of Database Systems*, 17, 1998.

[20] R. Fagin. Combining Fuzzy Information from Multiple Systems. *Journal of Computer and System Sciences*, 58:83–99, 1999.

[21] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *Proc. Twentieth ACM Symposium on Principles of Database Systems*, pages 102–113, 2001.

[22] H. P. Frei and P. Schäuble. Determining the effectiveness of retrieval algorithms. *Information Processing & Management*, 27(2):153–164, 1991.

[23] J. Gaoss and A. Ralescu. Adapting query representation to improve retrieval in a fuzzy database. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 3(1):57–77, 1995.

[24] T. G. Glagowski, K. L. Jones, and M. E. Stafford. A New Method for Implementing Fuzzy Retrieval from a Spatial Database. *Information Sciences*, 88:209–225, 1996.

[25] I. Glöckner. A Framework for Evaluating Approaches to Fuzzy Quantification. Technical report, Technische Fakultät, Universität Bielefeld, Postfach 100131, 33501 Bielefeld, 1999.

[26] I. Glöckner and A. Knoll. Fuzzy Quantifiers for Processing Natural Language Queries in Content-Based Multimedia Retrieval Systems. Technical report, Technische Fakultät, Universität Bielefeld, Postfach 100131, 33501 Bielefeld, 1997.

[27] I. Glöckner and A. Knoll. A Framework for Evaluating Fusion Operators Based on the Theory of Generalized Quantifiers. Technical report, Technische Fakultät, Universität Bielefeld, Postfach 100131, 33501 Bielefeld, 1998.

[28] I. Glöckner and A. Knoll. Architecture and retrieval methods of a search assistant for scientific libraries. In R. Decker and W. Gaul, editors, *Classification and Information Processing at the Turn of the Millenium*. Springer, 2000.

[29] I. Glöckner, A. Knoll, and A. Wolfram. Data Fusion Based on Fuzzy Quantifiers. Technical report, Technische Fakultät, Universität Bielefeld, Postfach 100131, 33501 Bielefeld, 1998.

[30] M. Gordon. Probabilistic and genetic algorithms for document retrieval. *Communications of the ACM*, 31(10):1208–1218, 1988.

[31] D. Harman. Overview over the second text retrieval conference (TREC-2). *Information Processing & Management*, 31(3):271–289, 1995.

[32] D. Harman. The second text retrieval conference (TREC-2). *Information Processing & Management*, 31(3):269–270, 1995.

[33] J. Helms, D. C. Neale, P. L. Isenhour, and J. M. Carroll. Data logging: High-level capturing and multi-level abstracting of user activities. *Proceedings of the 40th annual meeting of the Human Factors and Ergonomics Society*, 2000.

[34] I. R. Homann. Extraktion domänenspezifischer Informationen aus WWW-Sites. Diplomarbeit, Universität Bielefeld, Technische Fakultät, Postfach 100131, 33501 Bielefeld, 2000.

[35] A. L. Houston, H. Chen, B. R. Schatz, S. M. Hubbard, R. R. Sewell, and T. D. Ng. Exploring the use of concept spaces to improve medical information retrieval. *Descision Support Systems*, 30:171–186, 2000.

[36] HTML. http://www.w3.org/MarkUp/.

[37] HTTP. http://www.w3.org/Protocols/.

[38] L. Huang, M. Hemmje, and E. J. Neuhold. Admire: An adaptive data model for meta search engines. *Computer Networks*, 33:431–448, 2000.

[39] S. C. Hui and A. Goh. Incorporating fuzzy logic with neural networks for document retrieval. *Engng. Applic. Artif. Intell.*, 9(5):551–560, 1996.

[40] Java. http://java.sun.com/.

[41] M. F. Jiang, S. S. Tseng, and C. J. Tsai. Intelligent query agent for structural document databases. *Expert Systems with Applications*, 17:105–113, 1999.

[42] J. Kahlert and H. Frank. *Fuzzy-Logik und Fuzzy-Control*. Vieweg, Braunschweig, 2 edition, 1994.

[43] J. Kalinski. Text-Retrieval mit einem relationalen Datenbank-Management-System. *Informatik Forschung und Entwicklung*, 14:36–45, 1999.

[44] E. M. Keen. Presenting results of experimental retrieval comparisons. *Information Processing & Management*, 28(4):491–502, 1992.

[45] C. S. G. Khoo, S. H. Myaeng, and R. N. Oddy. Using cause-effect relations in text to improve information retrieval precision. *Information Processing and Management*, 37:119–145, 2001.

[46] M. H. Kim, J. H. Lee, and Y. J. Lee. Analysis of fuzzy operators for high quality information retrieval. *Information Processing Letters*, 46:251–256, 1993.

[47] D. H. Kraft and Frederick E. Petry. Fuzzy information systems: Managing uncertainty in databases and information retrieval systems. *Fuzzy sets and systems*, 90:183–191, 1997.

[48] J. H. Lee, W. Y. Kim, M. H. Kim, and Y. J. Lee. The effectiveness of fuzzy operators in information retrieval. *Fuzzy Logic and its Applications, Information Sciences, and Intelligent Systems*, pages 397–405, 1995.

[49] Linux. http://www.linux.org/.

[50] R. M. Losee. Upper bounds for retrieval performance and their use measuring performance and generating optimal boolean queries: Can it get any better than this? *Information Processing & Management*, 30(2):193–203, 1994.

[51] R. M. Losee. Determining information retrieval and filtering performance without experimentation. *Information Processing & Management*, 31(4):555–572, 1995.

[52] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Trans. Inf. Syst.*, 6(3):187–214, 1988.

[53] M. Ohta, A. Takasu, and J. Adachi. Reduction of expanded search terms for fuzzy English-text retrieval. *International Journal of Digital Libraries*, 3:140–151, 2000.

[54] OQL. http://www.ansi.org/oql.

[55] E. G. M. Petrakis and K. Tzeras. Similarity searching in the CORDIS text database. *Software - Practice and Experience*, 30:1447–1464, 2000.

[56] U. Pfeifer and N. Fuhr. Efficient processing of vague queries using a data stream approach. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. (Special Issue of the SIGIR Forum)*, pages 189–197. ACM Press, 1995.

[57] Rechercheassistent. http://www.ub.uni-bielefeld.de, 2000.

[58] D. E. Rose and R. K. Belew. A connectionist and symbolic hybrid for improving legal research. *International Journal Man-Machine Studies*, 35(1):1–33, 1991.

[59] G. Salton. The state of retrieval system evaluation. *Information Processing & Management*, 28(4):441–449, 1992.

[60] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[61] G. Salton, E. A. Fox, and H. Wu. Extended Boolean Information Retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[62] G. Salton and M. J. McGill. *Information Retrieval - Grundlegendes für Informationswissenschaftler*. McGraw-Hill, Hamburg, 1987.

[63] SQL. http://www.ansi.org/sql.

[64] P. Srinivasan, M. E. Ruiz, D. H. Kraft, and J. Chen. Vocabulary mining for information retrieval: rough sets and fuzzy sets. *Information Processing and Management*, 37:15–38, 2001.

[65] J. Tague-Sutcliffe. The pragmatics of information retrieval experimentation, revisited. *Information Processing & Management*, 28(4):467–490, 1992.

[66] Text Retrieval Conference. http://trec.nist.gov/.

[67] J. van Benthem. Questions About Quantifiers. *The Journal of Symbolic Logic*, 49(2):443–466, 1984.

[68] W. G. Waller and D. H. Kraft. A mathematical model of a weighted boolean retrieval system. *Information Processing & Management*, 15:235–245, 1979.

[69] F. Wiesman, A. Hasman, and H.J. van den Herik. Information retrieval: an overview of system characteristics. *International Journal of Medical Informatics*, 47:5–26, 1997.

[70] R. R. Yager. Families of OWA operators. *Fuzzy sets and systems*, 59:125–148, 1993.

[71] A. Yazici and D. Cibiceli. An access structure for similarity-based fuzzy databases. *Information Sciences*, 115:137–163, 1999.

[72] L. A. Zadeh. A Computational Approach To Fuzzy Quantifiers In Natural Languages. *Comp. & Maths. with Appls.*, 9(1):149–184, 1983.

[73] H.-J. Zimmermann. *Fuzzy Set Theory - And its Applications*. Kluwer Academic Publishers, 3 edition, 1996.