Ranked Information Retrieval on XML Data

Seminarausarbeitung

Referenten:
Bernadette Blum, Christian Nicolaus, Markus Uhl

Betreuer:
Stefan Siersdorfer

Dr. Ralf Schenkel
Universität des Saarlandes

15. Juli 2003

Inhaltsverzeichnis

1 Thema

Die Extensible Markup Language (XML) ist ein vom W3C eingeführter Standard zur Dokumentenauszeichnung. Diese Sprache definiert eine generische Syntax, mit deren Hilfe sich Daten mit einfachen, von Menschen lesbaren Tags auszeichnen lassen. XML ist flexibel genug, um für die unterschiedlichsten Problemfelder wie Webseiten, elektronischen Datenaustausch, Vektorgrafiken, Immobilienverzeichnisse und vieles mehr genutzt werden zu können.

Für XML Dokumente wurden einige Sprachen zur Anfrage und Transformation entwickelt, z.B. der W3C Standard XQuery, XML-QL, XQL oder Quilt. Diese Sprachen haben jedoch eines gemeinsam: sie unterstützen kein Information Retrieval, d.h. Anfragen mit nach ihrer Relevanz geordneten Rückgabewerten (Ähnlichkeitssuche).

Wir stellen zwei Anfragesprachen vor, die IR Anfragen an XML Dokumente erlauben: Zunächst ELIXIR und anschließend XRANK. Diese Anfragesprachen unterscheiden sich allerdings fundamental in ihren zugrundeliegenden Konzepten.

Während ELIXIR eine zusätzliche IR Funktionalität für eine bereits existierende SQL-ähnliche XML Anfragesprache implementiert, stellt XRANK ein eigenes System zur Verfügung, das Schlüsselwort-basierte Ähnlichkeitssuche nicht nur auf XML Daten, sondern auch auf HTML Dokumenten ermöglicht.

Im Folgenden werden wir zunächst eine Einführung in für die weiteren Ausführungen grundlegende Konzepte des Information Retrieval geben und anschließend allgemein auf Information Retrieval bei XML Daten eingehen. Danach stellen wir ELIXIR vor, gefolgt von XRANK. Zum Schluss fassen wir beide Verfahren noch einmal zusammen und gehen auf die Unterschiede und Gemeinsamkeiten ein.

2 Einführung in Information Retrieval

Definition: Information Retrieval (IR) ist die Technologie zum Suchen in Sammlungen (Intranet, Internet ...) schwach strukturierter Dokumente, z.B. Text, HTML, XML.

Anwendungsgebiete des IR sind u.a. Suchmaschinen (z.B. Google), digitale Bibliotheken oder Ähnlichkeitssuche in wissenschaftlichen Daten.

Die Daten bzw. Dokumente werden normalerweise durch ihre Eigenschaften (features) repräsentiert. Dies sind bei Textdokumenten vor allem der Text selbst sowie die Struktur der Dokumente.

Grundlegende Technik der *Textanalyse* ist das Vektorraummodell. Es basiert auf den Häufigkeiten der in den Dokumenten vorkommenden Terme. Terme können u.a. Wörter, Sätze oder Wortstämme (morphologisch abgeleitete Präfixe) sein. Ausgehend von einer Menge T an Termen, werden die Dokumente als Vektoren d in einem hoch-dimensionalen Vektorraum $\Re^{|T|}$ dargestellt. Die einzelnen Komponenten d^t dieser Vektoren sind Werte für die Häufigkeit des Vorkommens eines bestimmten Terms $t \in T$ in einem Dokument.
Eine Anfrage q an eine Sammlung von Dokumenten wird analog zu den Dokumenten selbst auch als Vektor ausgedrückt.
Somit kann man die Ähnlichkeit zwischen einer Anfrage q und den Dokumenten d_i mithilfe einer Ähnlichkeitsfunktion berechnen, die jeweils den Abstand zwischen Anfragevektor und Dokumentvektoren misst (Abstandsfunktion). Je geringer der Abstand, desto ähnlicher sind sich Anfrage und Dokument, desto höher ist die Relevanz des Dokuments bei der Rückgabe der Ergebnisse.

Die *Strukturanalyse* dient hauptsächlich dazu, die Dokumente zusätzlich zu gewichten und somit die Reihenfolge der Ergebisse noch zu präzisieren. Dies geschieht z.B. durch eine Hyperlink-Analyse.
Das bekannteste Verfahren ist *Page Rank*, das bei Google eingesetzt wird. Es betrachtet das World Wide Web als gerichteten Graphen. Jedes Dokument (v.a. HTML-Seiten) wird durch einen Knoten repräsentiert und jeder Hyperlink zwischen Dokumenten durch eine gerichtete Kante. Bei der Berechnung des *page ranks* $r(q)$ einer Seite q wird von einem zufälligen Bewegen ("random walk") eines Web-Surfers ausgegangen. Mit einer Wahrscheinlichkeit von ϵ ist er zu der aktuellen Seite q "gesprungen" ("random jump"), d.h. er ist keinem Hyperlink gefolgt, und mit einer Wahrscheinlichkeit von $1-\epsilon$ hat er einen Hyperlink von einer Seite p zur aktuellen Seite q genutzt.
Dokumente mit einem höheren *page rank* werden entsprechend höher gewichtet.

Dies wird durch folgende Formel ausgedrückt, wobei n die Anzahl aller Webseiten und $outdegree(p)$ die Anzahl aller von p ausgehenden Hyperlinks angibt.

$$r(q) = \epsilon \cdot \frac{1}{n} + (1 - \epsilon) \cdot \sum_{(p,q) \in G} \frac{r(p)}{outdegree(p)}$$

3 Information Retrieval auf XML-Dokumenten

Während HTML Daten in (fast) unstrukturierter Form speichert, erlaubt es XML, Daten in einer zumindest semi-strukturierten Form zu speichern. Aus IR Sicht bedeutet diese "Strukturierung" einen großen Informationsgewinn, den man sich zunutze machen möchte.

Wie bereits in Kapitel 1 erwähnt, unterstützen die meisten XML Anfragesprachen allerdings keine IR Anfragen, d.h. sie bieten nicht die Möglichkeit, aufgrund von textlicher Ähnlichkeit nach Relevanz geordnete bzw. gewichtete Suchergebnisse zurückzuliefern. Allerdings gibt es neuerdings auch Erweiterungen dieser Sprachen, die eine entsprechende Suchfunktion zumindest teilweise unterstützen (z.B. XXL, XIRQL).

Die SQL-ähnliche Anfragesprache ELIXIR bietet darüber hinaus eine besondere Eigenschaft: Sie ermöglicht durch ihren Ähnlichkeitsoperator sogar textliche Vergleiche zwischen 2 Variablen. Dies ist in der Familie der XML Anfragesprachen offenbar einzigartig. Bisher war immer nur ein Vergleich zwischen einer Variablen und einer Konstanten (i.d.R. einem Schlüsselwort) möglich.

Die Besonderheit der Schlüsselwort-basierten Anfragesprache XRANK liegt darin, dass sie eine HTML-basierte Suchmaschine (à la Google) verallgemeinert. Mit XRANK kann man folglich Anfragen an XML Dokumente und darüber hinaus auch an HTML Dokumente sowie eine Mischung von beiden stellen. Liegt ein XML Dokument vor, so berücksichtigt XRANK die besonderen Eigenschaften (Semi-Struktur) von XML.
Bei der rein Schlüsselwort-basierten Suche auf XML Daten ist es nicht einfach zu entscheiden, welche(s) der vielen ineinandergeschachtelten Elemente zurückgegeben werden soll(en). XRANK entscheidet sich für das spezifischste Element, indem es das Element zurückgibt, das alle Schlüsselwörter enthält und das am "tiefsten" im - dem XML Dokument entsprechenden - Baum liegt, also das Element mit der höchsten Kontextinformation.
Darüber hinaus verwendet XRANK zur Berechnung der Relevanz eines Rückgabewertes zusätzlich die Struktur des XML Dokumentes, bestehend einerseits aus den "Hyperlinks" (Page Rank), andererseits aus der Hierarchie der Elemente (bezogen auf ihre Reihenfolge und Schachtelung).
Treten in einer Anfrage mehrere Schlüsselwörter auf, so berücksichtigt XRANK zusätzlich die Korrelation der Schlüsselwörter. Zum einen ist dies der Abstand zwischen den Schlüsselwörtern ("horizontal"). Zum anderen der Abstand zwischen den Schlüsselwörtern und dem XML Element, das als Ergebnis zurückgegeben wird ("vertikal").

4 ELIXIR

ELIXIR (expressive and efficient language for XML information retieval) ist eine Erweiterung der SQL-ähnlichen Anfragesprache XML-QL um einen Ähnlichkeitsoperator "∼", der genauso wie die gewöhnlichen XML-QL Operatoren verwendet werden kann. Allerdings können auf beiden Seiten dieses Operators Variablen stehen. Dies verleiht ELIXIR die besondere Ausdrucksfähigkeit, Ähnlichkeitsvergleiche auch zwischen zwei Variablen durchzuführen, welche allen anderen bekannten XML Anfragesprachen fehlt. Zur Berechnung dieser Ähnlichkeit wird WHIRL als Unterroutine aufgerufen. ELIXIR gibt die r besten Ergebnisse auf eine Anfrage zurück, wobei r ein Parameter ist, den der Benutzer beliebig bestimmen kann. ELIXIR ist in der Familie der Anfragesprachen wie folgt einzuordnen:

		data model	
		relational	XML
full support for	no	SQL, Datalog	Quilt, XML-QL, *etc.*
information retrieval?	yes	WHIRL	**ELIXIR**

Abbildung 1: ELIXIR in der Familie der Anfragesprachen

4.1 Die Anfragesprache ELIXIR

ELIXIR basiert auf der Anfragesprache XML-QL, die eine SQL-ähnliche Syntax besitzt. Eine ELIXIR-Anfrage (wie auch eine XML-QL-Anfrage) ist ein Ausdruck der Form:

```
CONSTRUCT C
WHERE     W in source.xml
```

Es folgt ein Beispiel für eine Anfrage an eine Datenbank *db.xml*, welche die Titel aller Bücher zurückliefert, die nach 1990 geschrieben wurden und deren Titel ähnlich den Titeln, der in *db.xml* gespeicherten CDs sind. Hier wird die besondere Ausdrucksfähigkeit von ELIXIR deutlich, indem der Ähnlichkeitsoperator "∼" zwei Variablen "vergleicht":

```
CONSTRUCT <item>$b</>
WHERE     <items.book year=$yb>$b</> in "db.xml",
          <items.cd>$c</> in "db.xml",
          $yb > 1990,
          $b ~ $c.
```

Die `CONSTRUCT` Klausel spezifiziert das gewünschte XML Ausgabeformat (hier alle Buchtitel als item-Elemente). Zeichen, denen ein "$" vorangestellt ist, sind Variablen. Das Beispiel enthält drei Variablen: $b, $c und $yb. Die Variablenbindungen werden in der `WHERE` Klausel generiert. W enthält Ausdrücke der Form "*Muster in Quelle*", wobei *Muster* ein Ausdruck ist, der festlegt, wie das XML Dokument zu durchlaufen ist und welche Teile folglich relevant sind. *Quelle* ist lediglich ein Verweis auf das XML Dokument, das durchsucht werden soll.
Während XML-QL geschachtelte Anfragen zulässt, ist dies in ELIXIR jedoch nicht möglich.

Zum besseren Verständnis des Ähnlichkeitsoperators "$\sim$" von ELIXIR gehen wir kurz auf WHIRL ein und erklären grundlegende Eigenschaften.

WHIRL (Word-based Heterogeneous Information Retrieval Logic)
WHIRL ist eine Erweiterung von Datalog um einen Ähnlichkeitsoperator "$\sim$" und kann nur relationale Daten verarbeiten. Der große Vorteil allerdings ist, dass WHIRL effizientes, geordnetes IR basierend auf textlicher Ähnlichkeit unterstützt.
Hier ein Beispiel einer WHIRL Anfrage, deren Syntax einer Horn-Klausel entspricht:

```
output($y, $a, $t) :- book($y, $a, $t), $y>1950, $t ~ $a.
```

Dabei spezifiziert der *Kopf* der Klausel die Ausgabe-Relation, welche die Variablenbindungen speichert. Der *Körper* der Klausel enthält zunächst die Eingaberelation und danach eine Konjunktion relationaler Prädikate.
In obigem Beispiel werden Bücher gesucht, die nach 1950 geschrieben wurden und deren Titel ähnlich ihren Autoren sind. Die Eingaberelation *book* besteht aus den Attributen Jahr ($y), Autor ($a) und Titel ($t), die Ausgaberelation *output* ebenfalls. Besonders hervorzuheben ist hier der Ähnlichkeitsoperator "$\sim$" von WHIRL, den der Algorithmus von ELIXIR verwendet.

Zur Ähnlichkeitsberechnung nutzt WHIRL die geläufigen IR Term-Vektor-Techniken (vgl. Kapitel 2). Zur Gewichtung der Terme benutzt WHIRL das TF-IDF Verfahren (term-frequency inverse-document-frequency). Die Termhäufigkeit $TF(d,t)$ eines Terms t in einem Dokument d ist die Anzahl von t in d. Die Dokumenthäufigkeit $DF(t)$ ist die Anzahl der Dokumente, die den Term t enthalten.
Der Wert des Dokumentenvektors eines Dokuments d an der Stelle t wird so berechnet:

$$d_t = -log(TF(d,t)+1) \cdot log(DF(t))$$

Die Ähnlichkeit einer Anfrage q und eines Dokuments d wird in WHIRL über das sog. Kosinus-Maß definiert, wobei T die Menge aller Terme t bezeichnet:

$$sim(q,d) = \sum_{t \in T} \frac{q_t \cdot d_t}{\|q\| \cdot \|d\|}$$

WHIRL berechnet darüber hinaus nicht das vollständige Ergebnis einer Anfrage, sondern nur die r besten Werte. Für tiefergehende Beschreibungen von WHIRL empfehlen wir einen Blick in das Literaturverzeichnis.

4.2 Der Anfragealgorithmus von ELIXIR

Betrachten wir zunächst ein

Naives Beispiel: Wir haben folgendes XML Dokument, *db.xml*:

```
<items><book>Traditional Ukrainian cookery</>
       <book>Being and nothingness</>
       <book>Shooting Elvis</>
       <cd>Ukrainian folk music</>
       <cd>Being there</>
       <cd>Milk cow blues</></>
```

Wir wollen nun alle Bücher finden, deren Titel ähnlich den Titeln der in dem XML Dokument gespeicherten CDs sind, und stellen folgende Anfrage Q_1:

```
<results> {
CONSTRUCT <item>$b</>
WHERE     <items.book>$b</> in "db.xml",
          <items.cd>$c</> in "db.xml",
          $b ~ $c
} </>
```

Ein naiver Algorithmus würde folgende Zwischenanfrage Q_2 generieren, um die Daten in eine relationale Form zu bringen, damit anschließend die Ähnlichkeitsberechung für jedes Tupel durchgeführt werden kann.

```
<q2> {
CONSTRUCT <tuple><b>$b</><c>$c</></>
WHERE     <items.book>$b</> in "db.xml",
          <items.cd>$c</> in "db.xml"
} </>
```

Hierbei wird deutlich, dass die Anwendung des Ähnlichkeitsoperators auf 2 Variablen gewichtige Probleme nach sich zieht, da die Berechnung des vollständigen Kreuzproduktes der Variablenbindungen mit anschließender vollständiger Ähnlichkeitsberechnung für alle Tupel äußerst zeitaufwändig wäre.

Abbildung 2 zeigt diesen Zusammenhang schematisch:

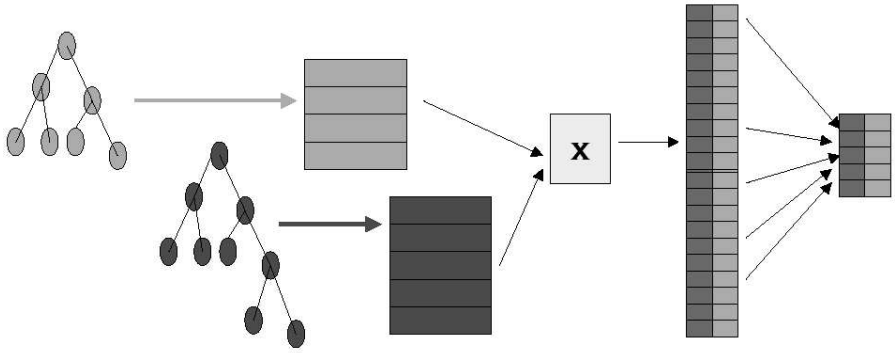

Abbildung 2: Ein naiver Algorithmus würde das vollständige Kreuzprodukt generieren

Um die Berechnung des vollständigen Kreuzproduktes und der vollständigen Ähnlichkeitswerte aller Tupel zu vermeiden, geht der Algorithmus von ELIXIR folgendermaßen vor:
Die XML Daten werden nicht einfach gänzlich in ein relationales Modell konvertiert, sondern es werden Zwischenschritte durch sog. Zwischenanfragen generiert. Dadurch wird die Berechnung des vollständigen Kreuzproduktes vermieden. Anschließend wird mithilfe von WHIRL die Ähnlichkeitsberechung für die r besten XML Dokumente durchgeführt. Diese Vorgehensweise ist entscheidend für die Effizienz des Algorithmus.
In Abbildung 3 wird dieses Vorgehen graphisch verdeutlicht. Die aufwändige Berechnung des vollständigen Kreuzproduktes ist entfallen.

Im Detail geht der ELIXIR Algorithmus dreistufig vor:

1. Zunächst wird die Ausgangsanfrage Q_1 in eine Menge von XML-QL Anfragen Q_2^i zerlegt, in denen die gewöhnlichen, von XML-QL stammenden, Prädikate (">", "<", "!=" ...) ausgewertet werden. Dabei geht der Algorithmus grundsätzlich so vor: Q_1 Muster-Ausdrücke werden in zwei verschiedenen Anfragen Q_2^i platziert, sofern sie Variablen enthalten, die über einen Ähnlichkeitsoperator verglichen werden. Dadurch wird vermieden, das vollständige Kreuzprodukt aller Variablen, die über einen Ähnlichkeitsoperator verglichen werden, berechnen zu müssen.

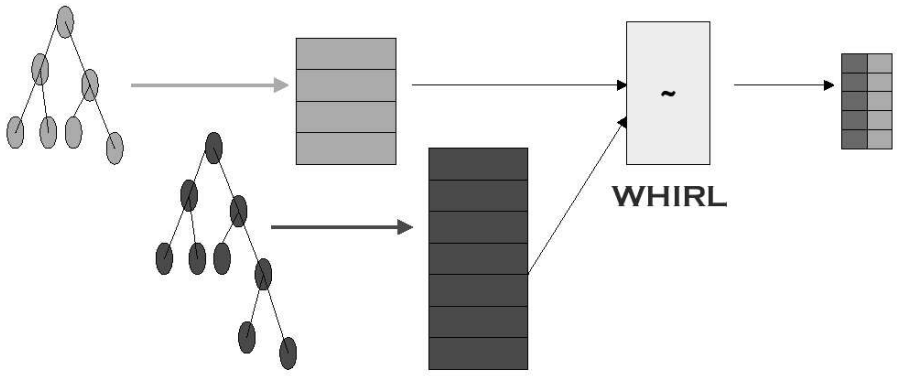

Abbildung 3: Die Vorgehensweise von ELIXIR

2. Anschließend generiert der Algorithmus eine WHIRL Anfrage Q_3, um die Ähnlichkeitsprädikate von Q_1 auszuwerten. Dazu verwendet Q_3 die Ausgaben der Anfragen Q_2^i als Eingaberelationen und berechnet eine nach fallender Relevanz geordnete Relation der r besten Ergebnisse.

3. Zum Schluss wird eine XML-QL Anfrage Q_4 generiert. Diese Anfrage konvertiert die durch Q_3 berechnete Relation in das durch die Q_1 `CONSTRUCT` Klausel gewünschte Ausgabeformat.

Beispiel: Wir haben wiederum folgendes XML Dokument, *db.xml*:

```
<items><book>Traditional Ukrainian cookery</>
       <book>Being and nothingness</>
       <book>Shooting Elvis</>
       <cd>Ukrainian folk music</>
       <cd>Being there</>
       <cd>Milk cow blues</></>
```

Erneut wollen wir alle Bücher finden, deren Titel ähnlich den Titeln der in dem XML Dokument gespeicherten CDs sind, und stellen erneut folgende Anfrage Q_1:

```
<results> {
CONSTRUCT <item>$b</>
WHERE     <items.book>$b</> in "db.xml",
          <items.cd>$c</> in "db.xml",
          $b ~ $c
} </>
```

Der Algorithmus zerlegt die Anfrage Q_1 gemäß dem oben genannten Grundsatz in zwei Zwischen-Anfragen Q_2^1 und Q_2^2, die zunächst alle Bücher ($b) und alle CD's ($c) separat selektieren:

```
<q21> {
CONSTRUCT <tuple><b>$b</></>
WHERE     <items.book>$b</> in "db.xml"
} </>

<q22> {
CONSTRUCT <tuple><c>$c</></>
WHERE     <items.cd>$c</> in "db.xml"
} </>
```

Die Ergebnisse der beiden Zwischenanfragen lauten wie folgt:

```
<q21><tuple><b>Traditional Ukrainian cookery</></>
     <tuple><b>Being and nothingness</></>
     <tuple><b>Shooting Elvis</></></>

<q22><tuple><c>Ukrainian folk music</></>
     <tuple><c>Being there</></>
     <tuple><c>Milk cow blues</></></>
```

Somit wurde vermieden, das vollständige Kreuzprodukt der Variablen $b und $c aufwändig zu berechnen.

Die WHIRL Anfrage Q_3 nutzt die Ausgabe der beiden vorhergehenden Anfragen als Eingaberelation und verarbeitet den Ähnlichkeitsoperator "$\sim$":

```
q3($b) :- q21($b), q22($c), $b ~ $c.
```

Q_3 berechnet die r besten Ergebnisse für die Ähnlichkeit von $b und $c mit folgendem Ergebnis:

```
<q3><tuple><b>Traditional Ukrainian cookery</></>
    <tuple><b>Being and nothingness</></></>
```

Abschließend bringt die XML-QL Anfrage Q_4 die durch Q_3 generierten Daten noch in die in Q_1 ursprünglich gewünschte Form:

```
<results> {
CONSTRUCT <item>$b</>
WHERE     <q3.tuple><b>$b</></> in "q3.xml"
} </>
```

Somit lautet das Ergebnis der Anfrageauswertung von Q_1 mit ELIXIR:

```
<results><item>Traditional Ukrainian cookery</>
         <item>Being and nothingness</></>
```

Abbildung 4 fasst das Vorgehen des Algorithmus graphisch zusammen.

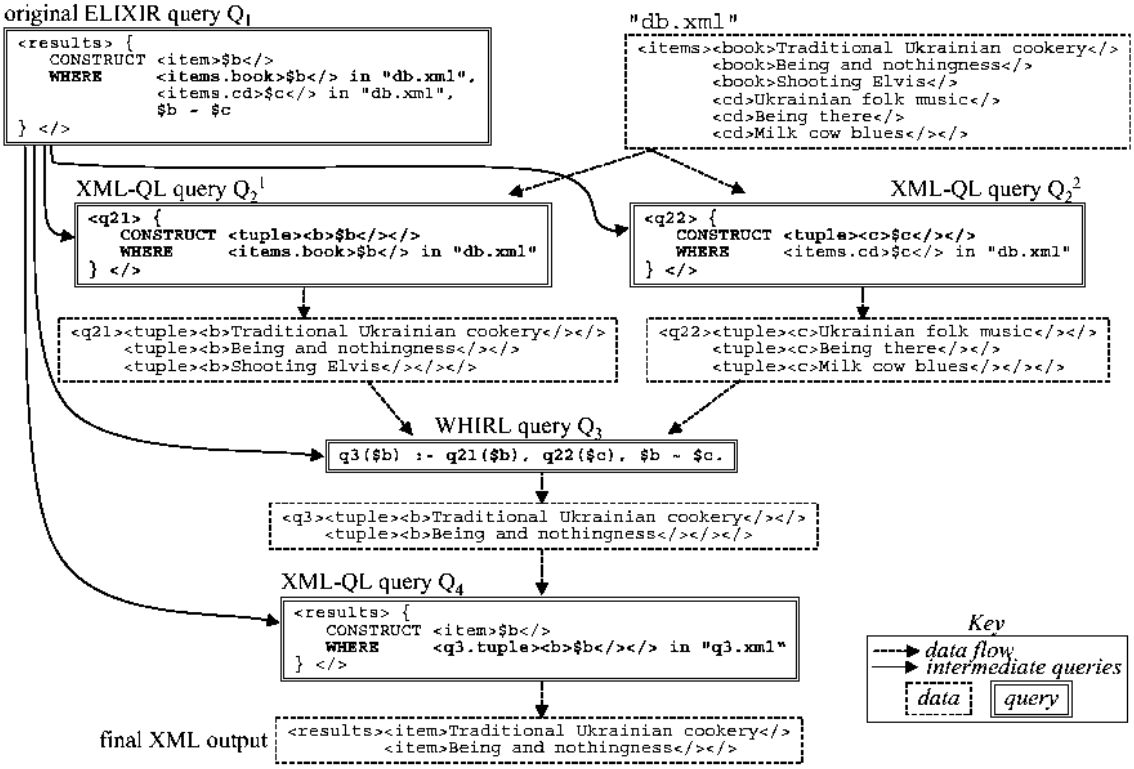

Abbildung 4: Der ELIXIR Algorithmus am Beispiel

4.3 Experimente

Zur experimentellen Auswertung wurden 13 Anfragen an 5 XML Datenbanken gestellt und ausgewertet. Dabei kam man kurz gefasst zu diesen Ergebnissen: Grundsätzlich ist die gesamte Laufzeit des Algorithmus abhängig von der Art der Anfrage und der Input-Daten. Darüber hinaus steigt sie nur marginal mit der Anzahl r der zurückgegebenen Antworten an. Ein linearer Zusammenhang wird mit der Zahl der Ähnlichkeitsprädikate ($\$v_i \sim \$v_j$) beobachtet. Dominierend ist im Allgemeinen die Generierung, Berechnung und Auswertung der Q_2^i Anfragen

in Schritt 1. Dies zeigt, dass das Parsen und Traversieren der Eingabedaten sehr zeitaufwändig ist, auch ohne die Berechnung von Ähnlichkeits-Prädikaten. Zusammengefasst ist der ELIXIR Algorithmus also auch effizient.

4.4 Zusammenfassung

In den vorhergehenden Abschnitten wurde ELIXIR vorgestellt. ELIXIR ist eine SQL-ähnliche Anfragesprache für XML Daten, die sogar Ähnlichkeitsberechnungen zwischen zwei Variablen durchführen kann. Dies ist ein Beleg für ihre *Ausdrucksstärke*. ELIXIR ist eine Erweiterung von XML-QL und nutzt zur Ähnlichkeitsberechnung den Operator "$\sim$" von WHIRL.
Der Algorithmus vermeidet die Berechnung des vollständigen Kreuzproduktes der Variablen, die über einen Ähnlichkeitsoperator verglichen werden, zunächst durch das Zerlegen der Ausgangsanfrage in mehrere Zwischenanfragen (eine Art "Filtern" der Ähnlichkeitsprädikate) und weiterhin durch das Aufrufen von WHIRL, welches die Ähnlichkeitsberechung schließlich durchführt und die r besten Ergebnisse zurückliefert.
Durch dieses Vorgehen ist ELIXIR auch *effizient*.
Allerdings können keine geschachtelten Anfragen (wie in XML-QL möglich) gestellt werden.
Zudem ist der strikte 3-stufige Ablauf des Algorithmus' in manchen Fällen nicht optimal (häufig, wenn mehr als ein Ähnlichkeitsoperator $\sim$ in einer Anfrage auftritt), was insbesondere die aufwendigen Berechnungen der Zerlegungsfunktion in Stufe 1 (Generierung der Zwischenanfragen Q_2^i) betrifft.
Darüber hinaus muss der Benutzer von ELIXIR die Struktur der Daten, die er durchsuchen möchte, kennen, da er sonst keine Anfrage stellen kann. Dies ist ein Nachteil der Struktur-basierten Anfragesprachen à la SQL. Im Gegensatz dazu benötigt der Benutzer bei Schlüsselwort-basierten Anfragesprachen dieses Wissen nicht.

5 XRANK

5.1 Einführung

XRANK ist ein System zur Auswertung von Anfragen an XML-Dokumente. Als Resultate werden nicht immer ganze Dokumente, sondern oft einzelne Tags zurückgeliefert. XML-Tags bezeichnen wir im Folgenden auch als XML-Elemente. Die Resultate der Anfragen werden in Form von Ranglisten zurückgegeben.
Um Anfragen an ein XML-Dokument zu stellen, muss man seine Struktur kennen. Entweder muss also der Nutzer, der eine Anfrage stellen möchte, Kenntnis über die zugrundeliegende Struktur der XML-Tags haben, oder alternativ kann die Struktur auch während der Auswertung der Anfrage erforscht werden. Da sich XRANK an der zweiten Lösungsmöglichkeit orientiert, ermöglicht dieses System eine Volltextsuche - Anfragen können im Google-Stil gestellt werden, ohne Kenntnis über den Aufbau der zu durchsuchenden Dokumente.

Die Resultate von Anfragen sollen nicht in beliebiger Reihenfolge, sondern nach deren Relevanz sortiert werden. Die besten Treffer sollen dabei an erster Stelle angezeigt werden.
Beim Ermitteln des Rankings sollen die folgenden Aspekte berücksichtigt werden:

- zugrundeliegende Hyperlink-Struktur der XML-Dokumente:
  Die hierarchische Struktur von XML-Dokumenten erlaubt sowohl Verweise innerhalb von Dokumenten als auch zwischen verschiedenen Dokumenten. Beim Berechnen des Rankings sollen diese beiden Arten von Hyperlinks beachtet werden.

- Spezifität der Resultate:
  Je tiefer ein Tag im Schachtelungsbaum sitzt, desto spezifischer ist es, da mit zunehmender Tiefe auch die Kontextinformation zunimmt. Zurückgeliefert wird das tiefstmöglichste Element im Baum, das alle gesuchten Schlüsselwörter beinhaltet.

- Abstand der gesuchten Begriffe:
  Nicht nur der Abstand aller Schlüsselwörter im Resultat voneinander wird betrachtet, sondern auch die Distanz des zurückgelieferten Elementes von jedem einzelnen Schlüsselwort. Dabei soll dieser errechnete Abstand möglichst gering sein.
  Achtung: Resultate mit geringem Abstand der Suchbegriffe können trotzdem unspezifisch sein!

5.2 XRANK-Architektur

Abbildung 5 gibt einen Überblick über den Aufbau und die Funktionsweise des XRANK-Systems. Für jedes in der Ansammlung von XML-Dokumenten

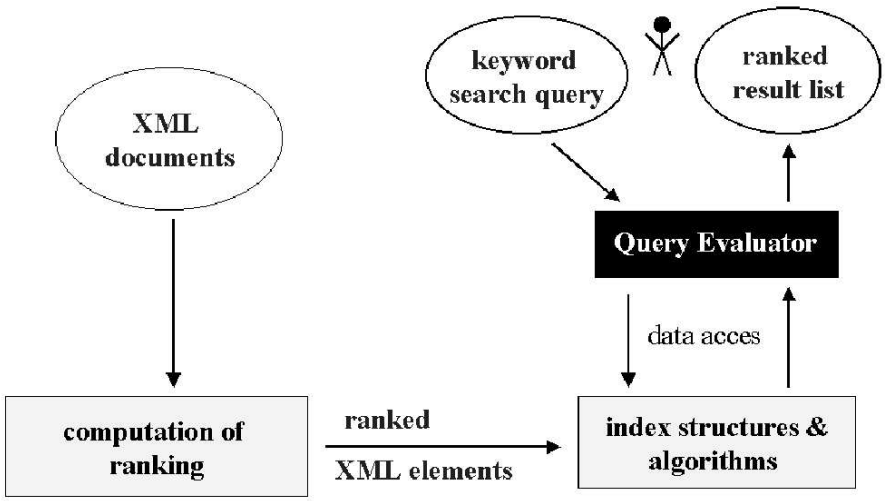

Abbildung 5: Architektur des XRANK-Systems

vorkommende XML-Element wird ein ElemRank (vgl. Abschnitt 5.4.1) berechnet und in einer angemessenen Indexstruktur (vgl. Abschnitt 5.5) aufgenommen. Der QueryEvaluator kann unter Benutzung dieser Indexstruktur eine Suchanfrage effizient verarbeiten und ein geranktes Suchergebnis ausgeben.

5.3 Datenmodell

Abbildung 6 zeigt ein XML-Dokument, bei dem es sich um eine Sammlung von CDs handelt. Dieses Dokument dient im Folgenden als Beispieldokument zur Erklärung von Datenstrukturen und Algorithmen. Die gesuchten Begriffe sind hier "R.E.M." und "Religion".

Eine Ansammlung von XML-Dokumenten wird als ein gerichteter XML-Graph $G = (V, CE, HE)$ betrachtet. V sei hierbei die Menge der XML-Elemente, CE die Menge der Kanten zwischen den Elementen (Containment-Kanten) und HE die Menge der Hyperlinks. $(u, v) \in CE$ gilt, wenn eine Kante von u nach v existiert. Das Prädikat $contains(v, k)$ ist wahr, wenn v das Wort k direkt oder indirekt beinhaltet.

Sei $Q = k_1, ..., k_n$ eine Suchanfrage und $R_0 = \{v | v \in V \wedge \forall k_i contains(v, k_i)\}$ die Menge aller Elemente, die alle gesuchten Worte beinhalten, dann ist die Ergebnismenge definiert durch die folgenden disjunkten Mengen:

1. $\{v | v \in R_0 \wedge \forall c(v, c) \in CE \Rightarrow c \notin R_0\}$
2. $\{v | \exists c((v, c) \in CE \wedge c \in R_0)$
   $\wedge \exists d, i((v, d) \in CE \wedge d \notin R_0 \wedge i \in N \wedge contains(d, k_i))\}$

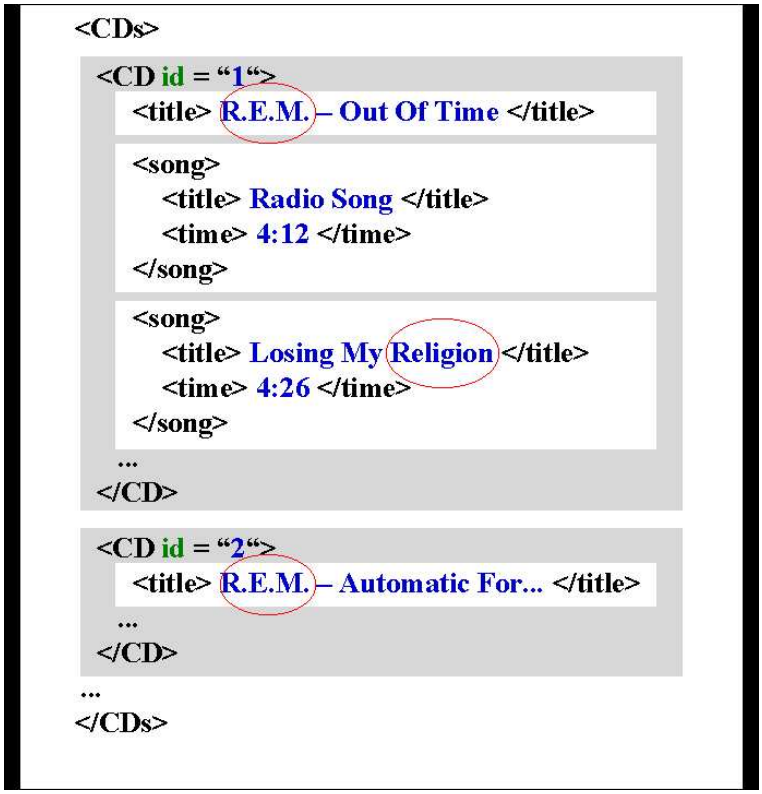

Abbildung 6: Beispieldokument

Die erste Menge umfasst alle XML-Elemente, die jedes gesuchte Wort beinhalten, aber keine Kinder besitzen, die ebenfalls jedes gesuchte Wort beinhalten. Damit wird sichergestellt, daß nur die spezifischsten Elemente zurückgeliefert werden. Die zweite Menge umfasst alle Elemente, die mindestens ein Kind besitzen, das jedes gesuchte Wort beinhaltet, und mindestens ein Kind besitzen, welches ein oder mehrere Suchbegriffe beinhaltet.

5.4 Ranking

5.4.1 ElemRank

ElemRank ist ein Maß für die Wichtigkeit eines XML-Elementes, das sowohl auf der Hyperlink-Struktur als auch auf der hierarchischen Strukur der XML-Dokumente basiert. Die Berechnung des ElemRanks baut auf dem PageRank-Verfahren von Google auf. Hierbei wird ein Random-Walk über den XML-Graphen modelliert und für jedes Element die Wahrscheinlichkeit ermittelt, dass der Surfer das Element besucht. Das Verfahren geht davon aus, dass ein Ele-

ment, welches über eine hohe Besuch-Wahrscheinlichkeit verfügt, ein wichtigeres Element ist als ein Element mit einer geringeren Besuch-Wahrscheinlichkeit.

Hält sich der Surfer bei einem Element u auf, kann er entweder mit der Wahrscheinlichkeit α über eine Hyperlink-Kante, mit der Wahrscheinlichkeit β über eine Containment-Kante oder mit der Wahrscheinlichkeit γ über eine Rückwärts-Containment-Kante zu einem anderen Element v navigieren. Außerdem besteht die Möglichkeit, dass der Surfer mit der Wahrscheinlichkeit $(1-\alpha-\beta-\gamma)$ einen direkten Sprung auf ein Element v macht.

Die Berechnung der Besuch-Wahrscheinlichkeit und somit des ElemRanks für ein Element v erfolgt mit der folgenden Formel:

$$\begin{aligned} e(v) &= (1-\alpha-\beta-\gamma) \times \frac{1}{N_e} \\ &\quad +\alpha \times \sum_{(u,v)\in HE} \frac{e(u)}{N_h(u)} \\ &\quad +\beta \times \sum_{(u,v)\in CE} \frac{e(u)}{N_c(u)} \\ &\quad +\gamma \times \sum_{(u,v)\in CE^{-1}} \frac{e(u)}{1} \end{aligned}$$

N_e ist die Anzahl aller in der Gesamtheit der XML-Dokumente vorhandenen Elemente. $N_h(u)$ und $N_c(u)$ beschreibt die Anzahl der von einem Element u abgehenden Hyperlinks bzw. Containment-Kanten. Die Menge CE^{-1} beschreibt die Menge aller Rückwärts-Containment-Kanten. Eine Rückwärtskante (u,v) besteht dann, wenn eine Containment-Kante (v,u) existiert.

Der erste Summand ist die Wahrscheinlichkeit, dass der Surfer einen direkten Sprung von einem Element u auf das betrachtete Element v macht. Der zweite Summand beschreibt die Wahrscheinlichkeit, dass er über ein Hyperlink zum Element v gelangt und der dritte Summand beschreibt die Wahrscheinlichkeit, dass über eine Containment-Kante zu dem betracheteten Element v navigiert wird. Der letzte Summand berücksichtigt die Wahrscheinlichkeit, dass über eine Rückwärts-Containment-Kante navigiert wird.

Die Parameter α, β, γ dienen dazu, die verschiedenen Kantenarten unterschiedlich zu gewichten. Durch das Definieren von Rückwärtskanten wird die Möglichkeit geschaffen, dass die Besuch-Wahrscheinlichkeit eines Elementes direkten Einfluß auf die Besuch-Wahrscheinlichkeit seines Parent-Elementes hat.

5.4.2 Ranking-Funktion

Um die Ergebnisse einer Suche über eine Menge von XML-Dokumenten adäquat zu ranken, ist eine Funktion gesucht, die die Hyperlink-Struktur der Dokumente, den Abstand der Suchworte sowie die Spezifität des Ergebnisses berücksichtigt.

Beispielsweise soll ein Element, welches alle Suchworte direkt beinhaltet höher gerankt werden, als ein Element, welches die Suchworte in verschiedenen Sub-Elementen beinhaltet.

Zunächst sei

$$r(v,k) = ElemRank(v_n) \times decay^{n-1}$$

mit $0 < decay < 1$ als eine Funktion definiert, die das Ranking für ein Element v bezüglich eines gesuchten Wortes k ermittelt. n bezeichnet die Länge des Pfades von Element v zu dem Element, das k direkt enthält. Hierbei wird der ElemRank von jenem Element v_n, welches das gesuchte Wort direkt beinhaltet, proportional zum Abstand des Ergebnisses v herunterskaliert. Damit wird gewährleistet, dass weniger spezifische Ergebnisse ein geringeres Ranking erhalten.

Bei oben genannter Funktion wird angenommen, daß das Suchwort k nur einmal in dem Element v vorkommt. Damit beim Berechnen des Rankings das mehrmalige Vorkommen des Wortes k berücksichtigt wird, wird die Funktion

$$r^*(v,k) = f(r_1, ..., r_m)$$

definiert. Hierbei wird bei m Vorkommen des Wortes k für jedes Vorkommen i mittels der Funktion $r(v,k)$ ein Ranking r_i berechnet. Die berechneten Werte r_i werden anschliessend mit einer Aggregations-Funktion $f = max$ oder $f = sum$ zusammengeführt.

Das Ranking für ein Element v bezüglich einer Suche Q mit den Suchworten $k_1, ..., k_n$ wird wie folgt berechnet:

$$R(v,Q) = (\sum_{1 \le i \le n} r^*(v,k_i)) \times p(v_1, k_1, ..., k_n)$$

Das Ranking setzt sich zusammen aus der Summe der Rankings bezügl. der gesuchten Worte k_i multipliziert mit einem Wert, der umgekehrt proportional zum Abstand der gesuchten Worte ist. Die Proximity $p(v\_1, k\_1, ..., k\_n)$ ist die Länge eines minimalen Fensters, welches alle Worte beinhaltet.

5.5 Datenstrukturen und Algorithmen

naiver Ansatz Als naive Datenstruktur kann man zur Auswertung von Anfragen die im Information Retrieval bereits bekannten invertierten Listen verwenden. Dabei wird zu jedem gesuchten Schlüsselwort eine Liste aller Dokumente - im Fall von XML-Dokumenten also aller *Elemente* - gespeichert, die dieses Schlüsselwort indirekt oder direkt enthalten. Resultate werden bestimmt, indem man auf alle diese Listen zu den verschiedenen Suchbegriffen einen Merge-Algorithmus anwendet, der alle Elemente liefert, die jeden gesuchten Begriff enthalten.
Im Fall von XML-Dokumenten ist dieses Vorgehen mit einem Nachteil behaftet: Während des Mergens der Listen werden alle Vater-Kind-Beziehungen zwischen

den XML-Elementen missachtet. Deshalb werden nicht nur wie in Abschnitt 5.3 beschrieben die spezifischsten Elemente zurückgeliefert, die alle Suchbegriffe enthalten, sondern auch alle ihre Ahnen.
Desweiteren benötigt man zum Speichern dieser Indexstrukturen aufgrund der redundanten Resultate zu viel Speicherplatz. Es werden nämlich auch alle im XML-Baum sehr weit oben stehenden Elemente in die invertierten Listen eingefügt, die wegen ihrer geringen Kontextinformation oft irrelevant für den Nutzer sind.
Auch das Ranking kann verfälscht werden, da spezifischere Resultate - Elemente, die in der Baumdarstellung von der Wurzel relativ weit entfernt sind - nicht tendenziell ein höheres Ranking zugewiesen bekommen als eher unspezifische Resultate. Eine hohe Spezifität eines Elementes sollte jedoch beim Ranking positiv ins Gewicht fallen, da der typische Nutzer von Suchmaschinen Resultate mit möglichst viel Kontextinformation bevorzugt.

Dewey IDs Um obiges Problem zu lösen, werden die einzelnen Elemente durch sogenannte Dewey IDs charakterisiert. Dabei werden Elemente mithilfe durch den Pfad von der Wurzel bis zum Element identifiziert. Je kürzer die Dewey ID eines Elementes ist, desto weiter oben im XML-Baum befindet es sich. Besteht die Dewey ID nur aus einer Zahl, so handelt es sich um die Dokument-ID und somit um ein ganzes XML-Dokument. XML-Dokumente sind stets Wurzeln von Bäumen. Allgemein ist die Dewey ID jedes Elementes ein Prefix der Dewey IDs aller seiner Nachkommen (Unterelemente (Kinder), Unterelemente von Unterelementen, usw.).

Stellt man das XML-Dokument aus Abbildung 6 als Baum dar, so dass jeder Knoten ein XML-Element darstellt, so kann man, wie in Abbildung 7 dargestellt, jeden Knoten mit seiner Dewey ID versehen.

5.5.1 DIL

DIL steht für Dewey Inverted List. So bezeichnet man die Datenstruktur, auf der der DIL-Algorithmus operiert. Eine Dewey Inverted List (DIL) zu einem Schlüsselwort enthält die Dewey IDs aller XML-Elemente, die dieses Schlüsselwort *direkt* beinhalten. Jede DIL ist aufsteigend nach Dewey IDs sortiert. Desweiteren gehören zu jeder Dewey ID das nach der in Abschnitt 5.4.2 vorgestellten Formel berechnete Ranking und eine Liste aller Positionen, in der das Schlüsselwort in dem betreffenden Element zu finden ist. Abbildung 8 zeigt die passenden DILs zu den in unserem Beispiel gesuchten Begriffen "R.E.M." und "Religion".

DIL-Algorithmus Die Kernidee des DIL-Algorithmus' ist das Bestimmen des längsten gemeinsamen Präfixes (longest common Präfix - LCP) von Dewey IDs. Zur Ausführung des Algorithmus' wird ein sogenannter Dewey Stack angelegt, der seinen Zustand bei jeder Iteration verändert. In diesem Dewey Stack wird die gerade betrachtete Dewey ID gespeichert; außerdem bei n verschiedenen Suchbegriffen jeweils n Spalten zum Speichern von Ranking und position

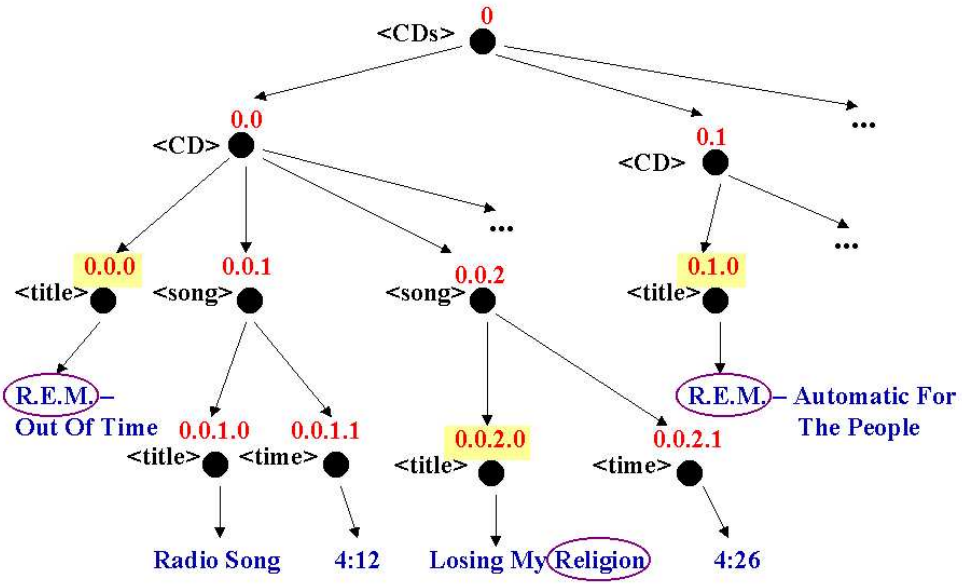

Abbildung 7: Dewey IDs

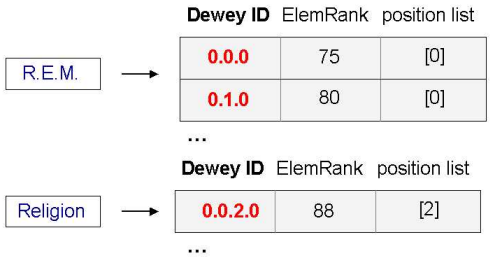

Abbildung 8: DILs für die gesuchten Begriffe

	DeweyID	rank [1]	rank [2]	posList [1]	posList [2]	pot_result
1.	**0**	75		0		y
	0	70		0		n
	0	65		0		n

Abbildung 9: DIL Algorithmus - Iteration 1

	DeweyID	rank [1]	rank [2]	posList [1]	posList [2]	pot_result
1.	**0**	75		0		y
	0	70		0		n
lcp	**0**	65		0		n

→

2.	DeweyID	rank [1]	rank [2]	posList [1]	posList [2]	pot_result
	0		88		2	n
	2		83		2	n
	0	70	78	0	2	y
	0	65	73	0	2	n

Abbildung 10: DIL Algorithmus - Iteration 2

List allokiert. Jeder Präfix der Dewey ID, einschließlich der Dewey ID selbst, verfügt über ein boolsches Flag, das angibt, ob das repräsentierte Element ein potentieller Kandidat für die Ergebnismenge ist.
Ein Element wird in die Ergebnismenge eingefügt, wenn es

1. als potentielles Resultat markiert worden ist und
2. alle Suchbegriffe enthält, d.h. für jeden Suchbegriff über einen Eintrag im Feld Position List verfügt.

Vor jeder Iteration wird aus allen DILs die minimale Dewey ID ausgewählt und auf einen Stapel gelegt. In unserem Beispiel wird also mit der ID 0.0.0 begonnen. Der LCP wird jeweils mit der Dewey ID gebildet, die vorher auf den Stack gelegt worden ist. Anschließend wird das Element, das durch diesen LCP repräsentiert wird, als potentielles Resultat markiert.
Da der Stack initial leer war, gibt es noch keine Dewey ID, mit der der LCP gebildet werden kann, deshalb ist der LCP leer. In diesem Fall wird das Element, das durch die gesamte Dewey ID repräsentiert wird, als potentielles Resultat markiert (in Abbildung 9 durch y gekennzeichnet). Allerdings beinhaltet das Element nur das Schlüsselwort "R.E.M.", für "Religion" liegt kein Position List-Eintrag vor. Deshalb wird es nicht in die Ergebnismenge eingefügt.
Durch erneute Extraktion des Minimums aller Dewey IDs wird die ID 0.0.2.0

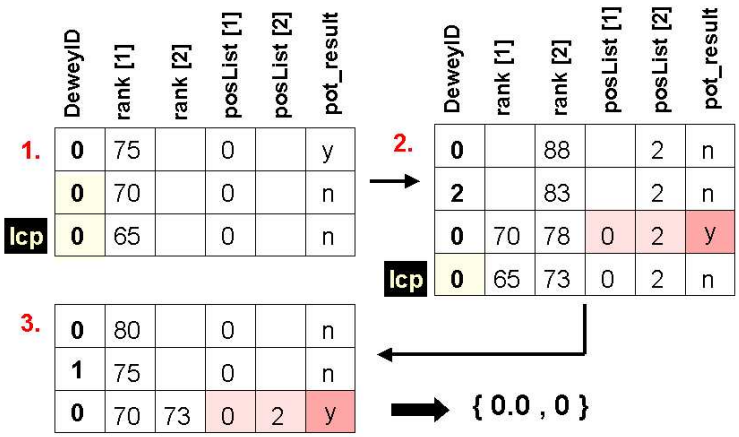

Abbildung 11: DIL Algorithmus - Iteration 3

ausgewählt. Bestimmt man nun LCP (0.0.0, 0.0.2.0) so erhält man 0.0. Die Position List-Einträge von 0 und 0.0 bleiben also für den Begriff "R.E.M." erhalten, der Eintrag von 0.0.0 wird gelöscht. Hinzugefügt werden nun die Ranking- und Position List-Einträge für die Elemente der IDs 0.0.2 und 0.0.2.0. Der LCP 0.0 wird als potentielles Resultat markiert. Wie man in Abbildung 10 sieht, verfügt diese ID auch über Position List-Einträge für beide Begriffe, d.h., dass beide Begriffe in dem dazugehörigen Element enthalten sind. Wegen der Bildung des LCPs ist das Element auch so spezifisch wie möglich. Die ID 0.0 wird also in die Resultatmenge eingefügt.
Nach der 3. Iteration, die durch Abbildung 11 veranschaulicht wird, befinden sich also die Elemente mit den Dewey IDs 0.0 und 0 in der Resultatmenge.
Der Algorithmus stoppt, nachdem er alle DILs systematisch abgearbeitet hat. Zur Ausgabe werden die Elemente der Ergebnismenge absteigend nach ihrem Ranking sortiert und ausgegeben. Wünscht der Nutzer nur die m treffendsten Resultate mit m kleiner n, so werden einfach nur die ersten m Elemente der sortierten Liste ausgegeben.

Da der Algorithmus alle DILs komplett abarbeitet, führt er womöglich zu viele Schritte aus, falls die gewünschte Anzahl an Resultaten sehr gering ist. Obwohl der Nutzer nur m Elemente in der Resultatliste sehen möchte, kann es sein, dass sich nach der Anwendung des DIL Algorithmus' viel mehr Elemente in der Ergebnismenge befinden, als zur Ausgabe benötigt werden.
Diese Tatsache führt zu der Idee, die invertierten Listen nicht aufsteigend nach Dewey IDs, sondern absteigend nach dem Ranking zu sortieren.

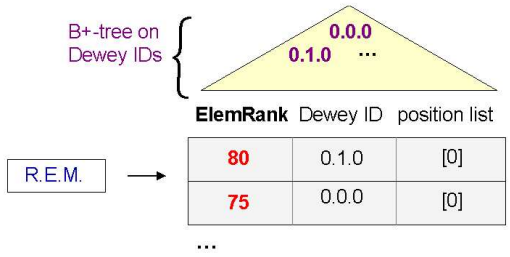

Abbildung 12: RDIL für den gesuchten Begriff "R.E.M."

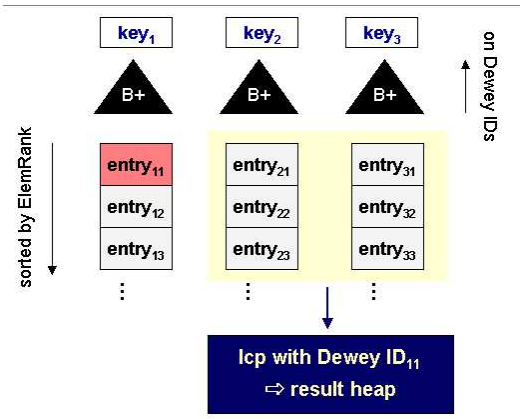

Abbildung 13: RDIL für den gesuchten Begriff "R.E.M."

5.5.2 RDIL

RDIL steht für Ranked Dewey Inverted List und greift die oben erwähnte Idee auf, die Listen nach dem Ranking zu sortieren. Um aber trotzdem noch effizient den LCP bestimmen zu können, wird zusätzlich zu der absteigend nach dem Ranking sortierten Liste noch ein mit den Dewey IDs gebildeter B+-Baum gespeichert. Da aber bei der Implementierung der Platz nicht zum Speichern eines separaten B+-Baums ausreicht, besitzt jede Dewey ID einen Eintrag, der ihre Position im B+-Baum spezifiziert. Die Spalten für Ranking-Einträge werden analog zu DIL angelegt.
Abbildung 12 zeigt die RDIL für den gesuchten Begriff "R.E.M.", für "Religion" lässt sich die RDIL analog bilden.

RDIL-Algorithmus Genau wie DIL basiert RDIL auf der Bildung des längsten gemeinsamen Präfixes zwischen Dewey IDs. Dabei wird der B+-Baum als Black Box verwendet, indem einfach seine Funktionalität zur LCP-Bildung be-

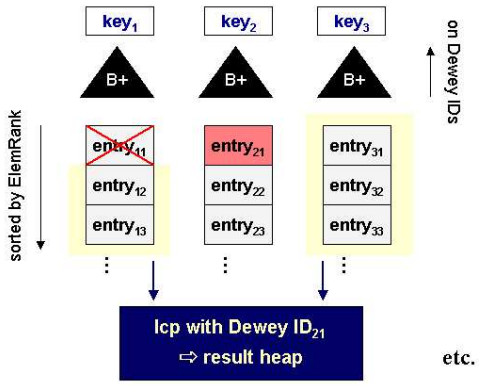

Abbildung 14: RDIL für den gesuchten Begriff "R.E.M."

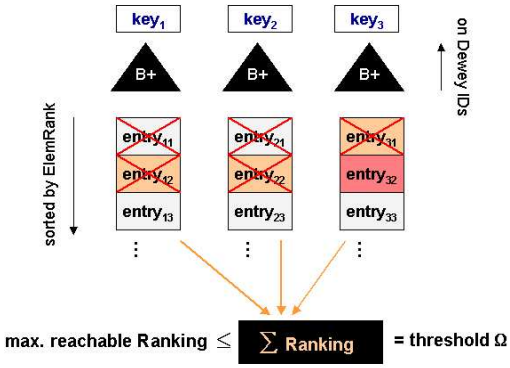

Abbildung 15: RDIL für den gesuchten Begriff "R.E.M."

nutzt wird. Diese Berechnung des LCP ist allerdings relativ teuer.
Im Gegensatz zur Extraktion der minimalen noch nicht abgearbeiteten Dewey ID bei DIL werden die Listen in sequentieller Reihenfolge ausgewählt. Aus jeder Liste wird immer genau der erste noch nicht abgearbeitete Eintrag entnommen, also der Eintrag mit dem größten noch verfügbaren Ranking zu dem entsprechenden Schlüsselwort.
Nun wird der LCP der Dewey ID, die zu dem gewählten Eintrag gehört, mit allen anderen noch nicht abgearbeiteten - also im ersten Schritt allen - Dewey IDs zu den Listen aller anderen Schlüsselwörter gebildet. Dieser LCP wird zusammen mit seinem Ranking in den Resultat-Heap eingefügt, der nach Ranking sortiert ist.
Abbildungen 13 und 14 zeigen die ersten beiden Iterationen des RDIL Algorithmus'. In der zweiten Iteration wird der in der ersten Iteration ausgewählte Eintrag nicht mehr beachtet, da die LCP-Bildung kommutativ ist. Entweder wurde also eine Übereinstimmung zwischen den beiden gewählten Einträgen erkannt und im Ergebnis beachtet, oder der LCP zwischen den beiden Einträgen war leer.

RDIL scannt nicht alle Listen systematisch ab, sondern stoppt, sobald eine bestimmte Schwelle überschritten ist. Diese Schwelle berechnet sich wie folgt:
Aus jeder Liste nimmt man den zuletzt abgearbeiteten Eintrag und summiert alle Rankings der ausgewählten Einträge auf. Das maximale Ranking, das beim Fortführen des Algorithmus' noch erreicht werden kann, ist höchstens so groß wie die oben beschriebene Summe. Denn die Listen sind ja absteigend nach dem Ranking sortiert, und bei mehreren Schlüsselwörtern wird über die einzelnen Rankings, wie in Abschnitt 5.4.2 vorgestellt, summiert. Wir nehmen an, dass der Nutzer genau m Resultate in der Ergebnisliste sehen möchte. Der Algorithmus bricht genau dann ab, wenn es in dem Heap schon m Elemente mit einem Ranking, das größer als die berechnete Schwelle ist, gibt. Denn dann gilt nach dem Transitivitätsgesetz, dass das maximal noch erreichbare Ranking kleiner gleich den m besten Rankings im Resultat-Heap ist.

5.6 Experimente

In Experimenten wurde unter anderem ein direkter Vergleich zwischen DIL und RDIL unter gleichen Bedingungen durchgeführt, dessen Ergebnis wird kurz darstellen werden.
Bei dem Performanz-Vergleich wurde die benötigte Zeit in Abhängigkeit von der Anzahl der gesuchten Begriffe gemessen.
Der Algorithmus wurde mit zwei verschiedenen Kategorien von Dokumenten ausgetestet:

- Dokumente mit *hoher* Korrelation der gesuchten Begriffe:
  Bei hoher Korrelation der gesuchten Begriffe hat RDIL eine bessere Laufzeit als DIL, da RDIL nicht alle Listen systematsich abscannt, sondern bei Überschreiten einer bestimmten Schwelle abbricht. Liegen die gesuchten

Begriffe nahe beinander, so liefert der B+-Baum-Algorithmus zur Bestimmung des LCP Ergebnisse mit relativ hohem Ranking zurück, so dass die Schwelle schnell erreicht ist und der Algorithmus abbricht. DIL hingegen scannt alle Listen komplett ab und macht somit einige redundante Schritte.

- Dokumente mit *niedriger* Korrelation der gesuchten Begriffe:
  Bei niedriger Korrelation der gesuchten Begriffe hat ist die Laufzeit von DIL besser als die Laufzeit von RDIL. Denn wenn die Suchbegriffe teilweise sogar in verschiedenen Dokumenten vorkommen, liefert die B+-Baum Operation den leeren LCP zurück, so dass die Schwelle nicht so schnell überschritten wird. Da B+-Baum-Operationen sehr teuer sind, hat DIL schneller alle Listen systematisch abgescannt und somit eine bessere Laufzeit.
  Bei nur einem gesuchten Schlüsselwort hat allerdings RDIL die bessere Laufzeit, da dann nur eine Liste existiert und somit keine Dewey IDs aus anderen Listen zur Bildung des LCP zur Verfügung stehen. In diesem Fall werden die Elemente der ersten Liste einfach nur nach Ranking sortiert ausgegeben.

5.7 Zusammenfassung

Sowohl DIL als auch RDIL stützt sich auf die Bestimmung des größten gemeinsamen Präfixes zwischen Dewey IDs. Die nachfolgende Tabelle enthält eine kurze Auflistung der Unterschiede zwischen den beiden Verfahren.

DIL	**RDIL**
invertierte Listen nach Dewey IDs sortiert (aufsteigend)	invertierte Lsiten nach ElemRank sortiert (absteigend)
Extraktion des Minimums aller verbleibenden Dewey IDs	sequentielle Auswahl der nächsten Liste und des nächsten Eintrags
arbeitet alle Listen komplett ab	stoppt bei Überschreitung einer bestimmten Schwelle
bei niedriger Korrelation der gesuchten Begriffe schneller als RDIL	bei hoher Korrelation der gesuchten Begriffe schneller als DIL

6 Fazit

Bei ELIXIR und XRANK handelt es sich um zwei verschiedene Ansätze zum Auswerten von Anfragen, denen XML-Dokumente zugrunde liegen.
Bei ELIXIR handelt es sich um eine Erweiterung von XML-QL, bei der es zusätzlich einen Ähnlichkeitsoperator $\sim$ gibt. Der Nutzer muss also den strukturellen Aufbau der XML-Dokumente kennen, auf denen die Anfrage ausgewertet werden soll und in seiner Anfrage diese Strukturkenntnisse verwenden. Dahingegen unterstützt das XRANK-System die sogenannte Schlüsselwortsuche, bei der der

Nutzer im Google-Stil die gesuchten Begriffe eingibt, wozu er keine Kenntnisse über die Struktur der XML-Dokumente benötigt.
ELIXIR ist sehr ähnlich zu der Anfragesprache SQL. Textuelle Ähnlichkeit kann zwischen zwei Variablen überprüft werden (wobei die XML-Tags die Variablen repräsentieren). Natürlich kann auch die Ähnlichkeit einer Variablen zu einer Konstanten überprüft werden. Im Gegensatz dazu beschränkt sich XRANK auf die Ähnlichkeitsüberprüfung zwischen einer Variablen und einer Konstanten (Schlüsselwort).
Ein neuer Ansatz bei XRANK ist die zweidimensionale Abstandsmessung von Schlüsselwörtern. Einerseits wird der Abstand eines Schlüsselwort zu den anderen gesuchten Begriffen betrachtet (Breite im Baum). Andererseits wird nach Bestimmung der Ergebnismenge auch der Abstand von allen Schlüsselwörten zu den Elementen, die als Resultate geliefert werden, berechnet. Diese zweidimensionale Abstandsinformation fließt in die Bestimmung des Rankings der Resultate ein.

Für die Zukunft ist XRANK wohl relevanter, da es eine HMTL-Suchmaschine verallgemeinert und zudem die benutzerfreundliche Schlüsselwort-basierte Suche unterstützt.

Literatur

[1] Taurai Tapiwar Chinenyanga, Nicholas Kushmerick. *An expressive and efficient language for XML information retrieval*, 2002.

[2] Elliotte R. Harold, W. Scott Means. *XML in a Nutshell*, 2nd Edition, O'Reilly, 2002.

[3] William W. Cohen. *The WHIRL Approach to Integration: An Overview.*

[4] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier and D. Suciu. *Querying XML Data*, Data Engineering Bulletin, 22(3): 10-18, 1999

[5] Lin Guo, Feng Shao, Shavdar Botev, Jayavel Shanmugasundaram. *XRANK: Ranked Keyword Search over XML Documents*, 2003.