Selectivity Estimation for XML Queries

Thomas Beer Mostafa Khabouze Christian Linz

21. Juni 2003

Zusammenfassung

Die Anzahl der Ergebnisse einer XML-Abfrage zu wissen (bzw. zu schätzen) ist im Bereich der XML-Query Verarbeitung in vielerlei Hinsicht von besonderer Bedeutung. Vor allem Abfrage-Optimierer (eng. query optimizer) sind auf exakte Resultate angewiesen, aber auch für den Endnutzer bringen genaue Resultatschätzungen einen enormen Vorteil. Dieser kann z.B. bei einer zuerst zu stark eingeschränkten Query die Suchkriterien weiter ausweiten, um somit mehr Ergebnisse zu erhalten.

1 Einführung

Abfragen in XML arbeiten mit sogenannten Pfadausdrücken (eng. path expressions) um durch die XML-Daten Struktur zu navigieren. Damit Query Optimizer eine Optimierung der Anfrage durchführen können, müssen sie die Anzahl der zugehörigen Pfade kennen.

Definition

Selektivität: Die Selektivität eines *einfachen Pfadausdrucks (engl. simple path expression)* ist die Summe der übereinstimmenden Pfade des XML-Baums und des Query-Pfades[1]
Die Selektivity des Pfadausdruck *(A/B/D)* in Abbildung 1 ist z.B. 2

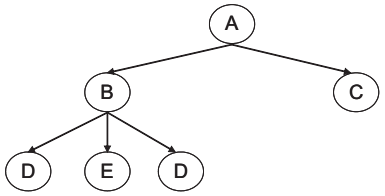


Abbildung 1: Beispiel, Definition Selektivität

Um die Selektivity von XML-Pfad-Ausdrücken zu bestimmen, benötigt man Statistiken über die Struktur der XML-Daten. Da diese Statistiken mehrmals je Optimiervorgang vom Optimierer benötigt (gelesen) werden, dürfen sie nicht zu umfangreich sein, um im Speicher gehalten werden zu können. Es geht hier nicht darum Speicherplatz zu sparen, sondern darum die Optimierungszeit möglichst gering zu halten. Dies stellt bei einzelnen XML-Bäumen kein großes Problem dar. Denkt man jedoch an Internet Applikationen, die tausende von XML-Bäumen vereinigen, so wird schnell klar, dass man nicht die komplette Struktur im Speicher halten kann. Deshalb müssen die zur Selekitvity Estimation verwendeten Statistiken zusammengefaßt (komprimiert) werden. In diesem Dokument werden Techniken vorgestellt, die die Struktur komplexer XML-Dokumente

[1] Der XML-Baum ergibt sich hier nicht aus einem einzelnen XML-Dokument, sondern vielmehr aus einer Vielzahl von XML-Dokumenten, wie sie z.B. bei Internet Applikationen benutzt werden

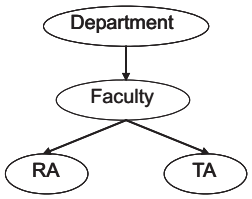


Abbildung 2: Join, Pattern Tree

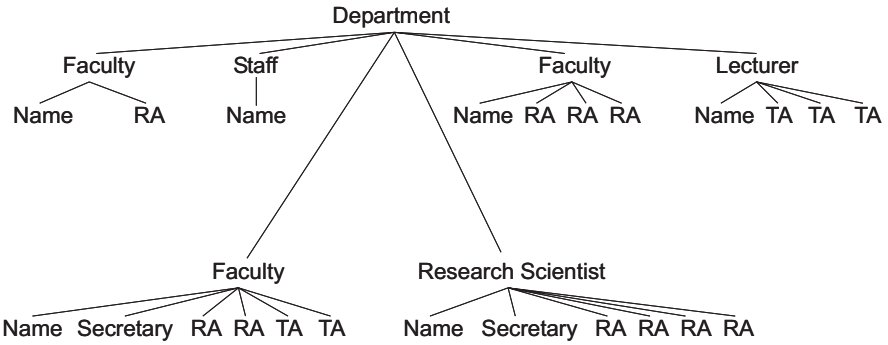


Abbildung 3: Beispiel XML Dokument, Baumstruktur

analyisieren und zusammenfassen. Zudem wird erklärt, wie man diese Statistiken dazu verwendet, die Selektivity von Pfadausdrücken zu berechnen.

Motivation:
Das Hauptanwendungsgebiet von "Selektivitätsabschätzung" ist das der Abfrage-Optimierer. Anhand des folgenden Beispiels wird veranschaulicht, warum es für diese wichtig ist, die Selektivität der Pfad-Ausdrücke zu kennen.
Gegeben sei der folgende XQuery Ausdruck:

FOR $f IN document ("personnel.xml")//department/faculty
WHERE count ($f/TA) > 0 AND count($f/RA) > 0
RETURN $f

Gesucht sind alle Fakultäten, die mindestens einen TA (Teaching Assistent) und einen RA (Research Assistent) haben. Wie auch bei relationalen Datenbanken ist auch hier die Reihenfolge der Join Ausführung von besonderer Wichtigkeit. Eine mögliche Auswertung dieses XQuery Statements könnte z.B. folgendemaßen aussehen:

Zuerst bestimmt man die Anzahl aller Department-, Faculty-, TA- und RA-Elemente. Danach werden drei Joins ausgeführt (für jede Kante ein Join, siehe Abbildung 2). Als Ergebnis erhält man alle entsprechenden "Teilbäume" des XML-Dokuments.

Annahme

- Anzahl der beteiligten Elemente ist bekannt
- Join Algorithmus: Nested Loop [2]

Element	Anzahl
Department	1
Faculty	3
TA	5
RA	10

Durchführung der Joins

1. Möglichkeit:

Join 1: (Faculty) - (TA)
Join 2: (Ergebnis von Join 1) - RA
Join 2: (Ergebnis von Join 2) - Department

Anzahl der benötigten Vergleiche:

Join 1: 3 * 5 = 15
Join 2: 1 * 10 = 10
Join 3: 1 * 1 = 1
Gesamt: 15 + 10 + 1 = 26

2. Möglichkeit:

Join 1: (Faculty) - (Department)
Join 2: (Ergebnis von Join 1) - RA
Join 2: (Ergebnis von Join 2) - TA

Anzahl der benötigten Vergleiche:

Join 1: 3 * 1 = 3
Join 2: 3 * 10 = 30
Join 3: 3 * 5 = 15
Gesamt: 3 + 30 + 15 = 48

Anhand dieses vereinfachten Beispiels ist leicht ersichtlich, warum es für Query Optimizer unerläßlich ist, die Anzahl der an der Query beteiligten Pfadausdrücke zu wissen. Aufgrund der geänderten Reihenfolge der Join Durchführung ergibt sich ein Unterschied von 22 (= 48-26) Vergleichen (die eingespart werden können, wenn man Informationen über die Häufigkeit des Auftretens bestimmter Pfade besitzt).

2 Path Trees

In diesem Abschnitt wird beschrieben, wie man die Struktur von XML Dokumenten mit Hilfe von Path Trees abbildet, diese zusammenfasst (komprimiert) und die Selektivitätsabschätzung (engl. Selectivity Estimation) vornimmt.
Ein Path Tree ist eine Baumstruktur, die die Struktur eines XML-Dokuments abbildet. Jeder Knoten des Path Trees repräsentiert dabei einen Pfad (ausgehend vom root-Element) und ist mit dem Namen des abgebildeten Elements beschriftet. Zudem wird auch noch die Anzahl dieser Elemente (die mit diesem Pfad erreicht werden können) mit angegeben (hier als *Frequenz* bezeichnet). Abbildung 4 zeigt ein Beispiel für einen Path Tree und das zugehörigen XML-Dokument.

[2] XQuery arbeitet mit einer Art nested loop join, für die genaue Implementierung siehe [Pfe00]

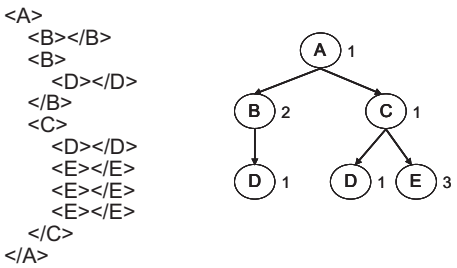


Abbildung 4: XML-Dokument und zugehöriger Path Tree

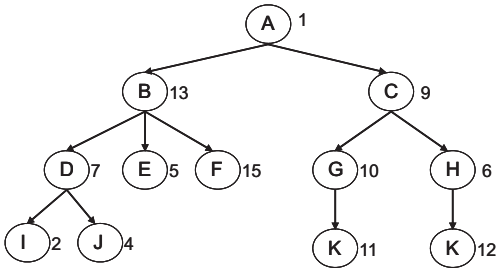


Abbildung 5: Ursprungs Path Tree

Erstellt wird der Path Tree mit Hilfe eines ereignisbasierten XML-Parsers (wie z.B. SAX). Wie bereits erwähnt, ist unter dem Begriff "XML-Dokument" eine Vielzahl von XML-Dokumenten zu verstehen. Um diese in einem Path Tree zusammenzufassen, führt man ein künstliches root-Element ein.
Ein Path Tree enthält alle zur Selektivitätsabschätzung benötigten Informationen. Auf Details der Selectivity Estimation wird später noch ausführlich eingegangen (siehe 2.5). Ein Problem das noch gelöst werden muss, ist der Umfang eines solchen Trees, denn dieser wird schnell größer, als der zur Verfügung stehende Speicher. Deshalb muss seine Struktur zusammengefaßt (komprimiert) werden.

Zur Komprimierung gibt es vier verschiedene Algorithmen:

- Sibling-*
- Level-*
- Global-*
- No-*

Die grundsätzliche Vorgehenweise ist bei allen vieren die gleiche: Man löscht die Knoten mit den niedrigsten Frequenzen und ersetzt sie durch Repräsentanten (sog. *-Knoten bzw. *-Nodes). Die vier Algorithmen unterscheiden sich darin, wieviel Information des ursprünglichen (nicht komprimierten) Path Trees erhalten bleibt.

2.1 Sibling-*

Der Name dieses Algorithmus (sibling=Geschwister) läßt bereits darauf schließen, hier werden "Geschwister-Knoten" vereinigt. Dazu markiert man wiederholt Knoten mit der niedrigsten Frequenz. Bei jeder Markierung prüft man, ob die Geschwister-Knoten bereits markierte oder *-Knoten enthalten. Ist dies der Fall, wird der aktuelle Knoten mit seinem Geschwister-Knoten zu

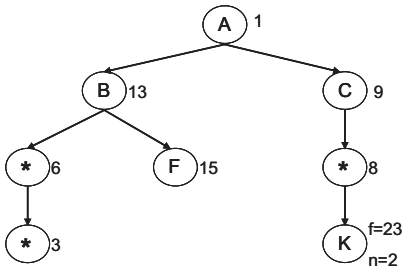


Abbildung 6: Sibling-*

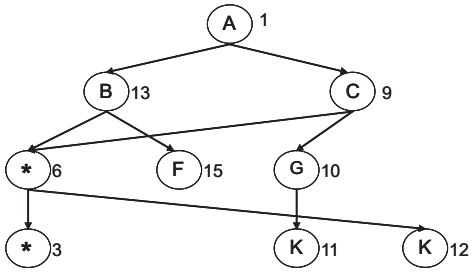


Abbildung 7: Level-*

einem *-Knoten "verschmolzen". Dadurch kann man die Größe des Path Trees beträchtlich verkleinern.
Der Vater einer *-Node wird zum Vater der gelöschten (und durch "*" ersetzten) Knoten; die Kinder dieser gelöschten Knoten werden Kinder der *-Node. Dadurch entstehen häufig Geschwister mit dem gleichen Namen (Label), welche wiederum verschmolzen werden. Da sowohl *-Knoten als auch "reguläre" Knoten als Repräsentant für gelöschte Knoten stehen können, enthalten alle Knoten:

- die Anzahl der gelöschten Knoten die sie repräsentieren
- die Gesamtfrequenz dieser Knoten (=Summe der Einzelfrequenzen)

Warum dies wichtig ist, wird später noch erläutert (siehe 2.5).
Wenn der Path Tree ausreichend zusammengefaßt (oder besser, verkleinert) wurde, durchläuft man den Baum und berechnet die durchschnittliche Frequenz der *-Knoten (= *Gesamtfrequenz / Anzahl*). Diese wird dann bei der Selectivity Estimation verwendet.

Anmerkung:
Der Sibling-Algorithmus versucht die genaue Position der gelöschten Knoten zu erhalten, dies geht jedoch zu Lasten des Speicherbedarfs (bzw. der Anzahl der benötigten Löschvorgänge um den Baum um n-Knoten zu reduzieren).
Um den Baum um n-Knoten zu reduzieren, müssen wir 2*n Knoten löschen.
Abbildung 5 und Abbildung 6 zeigen einen Path Tree und den zugehörigen Sibling-* Tree.

2.2 Level-*

Der Level-* Algorithmus funktioniert ähnlich wie der Sibling-Algorithmus. Im Vergleich dazu, werden hier alle zu löschenden Knoten eines Levels verschmolzen. Gelöscht werden, wie schon zuvor auch, die Knoten mit der niedrigsten Frequenz. Alle gelöschten Knoten eines Levels veschmilzt man in der *-Node dieses Levels. Wie auch beim Sibling-Algorithmus werden die Vater-Knoten Vater der *-Knoten (gleiches gilt für die Kinder-Knoten). Knoten mit gleichem Namen werden ebenfalls

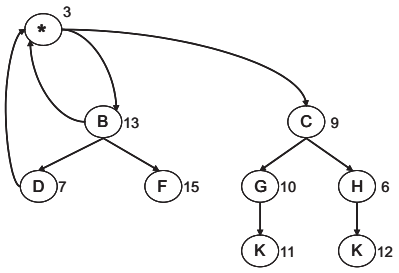


Abbildung 8: Global-*

vereinigt(Beispiel, siehe Abbildung 7)

Anmerkung:
Dieser Algorithmus erhält nur den Level der gelöschten Knoten, nicht wie im Sibling-Algorithmus die exakte Position. Deshalb müssen hier weniger Knoten gelöscht werden, um den Baum um die gleiche Anzahl (im Vergleich zum Sibling Algorithmus) von Knoten zu verringern. Um den Baum um n Knoten zu verringern, müssen weniger als $n+l$ Knoten gelöscht werden (l=Anzahl der Level).

2.3 Global-*

Bei diesem Algorithmus repräsentiert eine einzige *-Node alle gelöschten Knoten (natürlich werden wieder nur die Knoten mit der niedrigsten Frequenz gelöscht). Der Vater- und die Kinder-Knoten der gelöschten Knoten werden wiederum Vater bzw. Kinder der *-Node. Deshalb entsteht ein zyklischer Graph, wie er in Abbildung 8 zu sehen ist.

Anmerkung:
Da dieser Algorithmus keine Information über die Position der gelöschten Knoten erhält, müssen hier im Vergleich zu den beiden anderen Algorithmen am wenigsten Knoten gelöscht werden.
Um den Baum um n Knoten zu reduzieren, müssen n+1 Knoten gelöscht werden.

2.4 No-*

Der Name läßt bereits einiges erahnen. Dieser Algorithmus löscht die Knoten, ohne sie mit einem *-Node zu ersetzen. Der Path Tree wird zu "einem Wald mit vielen Wurzeln".
Der große Unterschied zu den anderen Algorithmen liegt in der Berechnung der Selektivität. Wird bei der No-* Methode ein Element des Pfadausdrucks nicht gefunden, wird angenommen, dass es den kompletten Pfadausdruck nicht gibt. Man geht davon aus, dass Knoten, die im zusammengefaßten Path Tree nicht existieren, auch im ursprünglichen Path Tree nicht vorkommen.
Im Vergleich dazu gehen die anderen Algorithmen davon aus, wenn ein Knoten im summierten Path Tree nicht gefunden wurde, dass er jedoch im ursprünglichen Path Tree existiert hat und durch eine *-Node ersetzt wurde.

Anmerkung:
Hier werden keine Ersetzungen der gelöschten Knoten vorgenommen, deshalb folgt daraus: Um den Baum um n Knoten zu reduzieren, müssen genau n Knoten gelöscht werden.

2.5 Selectivity Estimation

In diesem Abschnitt wird erläutert, wie man aus der zusammengefaßten Struktur der Path Trees die Selektivität berechnet. Die Vorgehensweise ist die folgende: Zuerst vergleicht man die Label der Knoten des XML Query Pfadausdrucks mit denjenigen des Path Trees um alle übereinstimmenden Pfade zu finden. Die Selektivität ergibt sich aus der Summe der Einzelfrequenzen der Knoten, zu denen der Pfadausdruck führt.

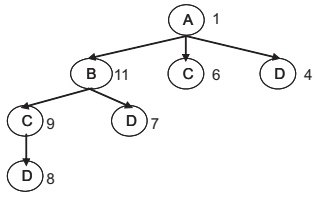


Abbildung 9: XML Baum Struktur

Path	Sel.	Path	Sel.
A	1	AC	6
B	11	AD	4
C	15	BC	9
D	19	BD	7
AB	11	CD	8

Abbildung 10: Markov Tabelle

Ausnahme: Endet ein übereinstimmender Query Pfadausdruck mit einem regulären Knoten (der vereinigte Knoten repräsentiert), wird geprüft, ob der zu diesem Knoten führende Pfad *-Nodes enthält. Ist dies der Fall, wird für die Selektivitätsabschätzung die durchschnittliche Frequenz (wie auch bei *-Nodes) verwendet. Anderenfalls verwendet man die Gesamtfrequenz dieses Knotens. Dies erklärt, warum reguläre Knoten (die vereinigte Nodes repräsentieren) immer beides, sowohl die Anzahl der gelöschten Knoten, als auch deren Gesamtfrequenz enthalten.
Kann man ein Element des gesuchten Pfad-Ausdrucks in den regulären (also den nicht *-Nodes) nicht finden, so überprüft man die *-Knoten auf eine mögliche Übereinstimmung. Sucht man z.B. nach folgendem Pfadausdruck *//A/B/C*, so werden folgende Übereinstimmungen gefunden: *//A/*/C, //A/*/*, //*/B/C* usw.
Wichtig: Pfade die nur aus *-Knoten bestehen, sind nicht erlaubt.
Wie bereits beim No-* Algorithmus erwähnt, geht man bei den *-Algorithmen davon aus, sobald man eine Übereinstimmung findet, die eine *-Node enthält, dass der ursprüngliche Path Tree diesen Ausdruck auch enthält. Es wird vielmehr sogar angenommen, dass alle Abfragen nach Pfaden suchen, die auch wirklich existieren.

3 Markov Tabellen

In diesem Abschnitt wird eine weitere Methode zur Abbildung einer XML-Daten Struktur vorgestellt, die sog. "Markov Tabellen". Eine Markov Tabelle ist in unserem Zusammenhang eine Tabelle, die bestimmte Pfade (des XML-Dokuments) bis zu einer durch "m" spezifizierten Länge und deren zugehörigen Selektivität enthält (siehe Abbildungen 9 und 10). Die Ordnung der Tabelle wird mit *(m-1)* angegeben [3]. Aus einer solchen Tabelle kann die Selektivität der einzelnen Pfade (jedoch nur der Länge zwei) sofort abgelesen werden. Im folgenden wird nun erläutert wie man die Selektivität längerer Pfade berechnen kann.
Die Selektivität längerer Pfadausdrücke kann durch Kombination von Pfaden der Länge m anhand der folgenden Formel berechnet werden:

$$f(t_1, t_2, \cdots, t_n) = \left(\prod_{i=1}^{n-1} P[t_i | t_{i+1}] \right) \cdot P[t_n] \cdot N \tag{1}$$

[3] In diesem Dokument werden nur Markov Tabellen der Ordnung eins, sogenannte *first-order Tabellen* betrachtet. Die hier vorgestellten Methoden können jedoch ohne weiteres auf Markov Tabellen höherer Ordnung angewendet werden.

$P[t_n]$	Wahrscheinlichkeit für das Auftreten eines Knoten mit dem Label t_n
N	Gesamtanzahl der Knoten
$P[t_i\|t_{i+1}]$	Wahrscheinlichkeit, dass vor einem Knoten mit dem Label t_{i+1} ein Knoten mit dem Label t_i steht

Erklärung der Formel:

Man geht bei diesem Ansatz davon aus, dass das Auftreten eines bestimmten Knoten innerhalb eines Pfadausdrucks nur von den $m-1$ "Vorgänger Knoten" abhängt. Man liest die Formel am besten von rechts nach links. Dabei ist dann $P[t_n] \cdot N$ der Erwartungswert für das Auftreten des Knoten t_n, und $P[t_{n-1}|t_n] \cdot P[t_n] \cdot N$ der Erwartungswert für das Auftreten des Knoten t_{n-1} vor dem Knoten t_n, usw.

$P[t_i] = \quad \frac{f(t_i)}{N}$

Weiter ist: $P[t_i|t_{i+1}] = \quad \frac{f(t_i,t_{i+1})}{f(t_{i+1})}$

$f(t_i, t_{i+1}) =$ Selektivität des Pfades t_i, t_{i+1}

Nimmt man diese Ersetzungen vor, kann man die vorherige Formel wie folgt schreiben:

$$f(t_1, t_2, \cdots, t_n) = \left(\prod_{i=1}^{n-2} \frac{f(t_i, t_{i+1})}{f(t_{i+1})} \right) \cdot f(t_{n-1}, t_n) \tag{2}$$

Zur Verdeutlichung noch ein kleines Beispiel: Gegeben ist die Markov-Tabelle (siehe (x)), n=3, m=2. Gesucht ist die Selektivität des Pfades $f(B, C, D)$

$$f(B, C, D) = \left(\prod_{i=1}^{3-2} \frac{f(B, C)}{f(C} \right) \cdot f(C, D) = \frac{9}{15} \cdot 8$$

3.1 Zusammenfassung von Markov Tabellen

Wie auch Path Trees werden Markov Tabellen noch weiter zusammengefaßt (komprimiert). Die drei dazu verwendeten Algorithmen werden im folgenden Abschnitt vorgestellt. Die Vorgehensweise ist ähnlich wie bei der Path-Tree-Variante. Die Pfade mit der niedrigsten Selektivität werden gelöscht.

Suffix-*
Dieser Algorithmus nutzt zwei spezielle **-Pfade*, "*" und "*/*". Dabei steht * für alle gelöschten Pfade der Länge eins, und */* für alle Pfade der Länge zwei. Löscht man einen Pfad der Länge 1 wird er sofort zum *-Pfad hinzugefügt. Beim */* Pfad hingegen werden die gelöschten Pfade nicht sofort hinzugefügt. Man merkt sich die gelöschten Pfade (bezeichnet mit S_D) und vergleicht bei jedem Löschvorgang den Startknoten des aktuellen Pfaden mit S_D. Enthält S_D einen Pfad der mit dem gleichen Knoten beginnt, vereinigt man diese beiden Pfade.

Beispiel: $S_D = \{(A, B)\}$, gelöscht wird $(A, C) \Longrightarrow$Vereinigung zu $(A, *)$

Die am Ende der Zusammenfassung noch in S_D stehenden Pfade werden zum */* Pfad hinzugefügt.

Global-*
Im Unterschied zum Suffix-* Algorithmus werden hier auch die gelöschten Pfade der Länge 2 sofort zum */*-Pfad hinzugefügt.

No-*
Ähnlich wie No-* bei Path Trees, werden auch hier die zu löschenden Pfade einfach gelöscht, ohne sie zu ersetzen. Bei der Berechnung der Selektivität wird deshalb die Selektivität eines nicht gefundenen Pfades mit Null bewertet. Genau wie beim No-* Algorithmus bei Path Trees ist auch hier die Annahme, wenn ein Pfad in der zusammengefaßten Markov Tabelle nicht gefunden wurde, dass er auch in der ursprünglichen Markov Tabelle nicht existiert hat.

3.2 Berechnung der Selektivity

Die Selektivität kann entweder direkt aus der Markov Tabelle abgelesen oder mit Hilfe der oben beschriebenen Formel (2) berechnet werden. Wird ein Pfad der Länge 1 nicht gefunden, verwendet man zur Berechnung die Selektivität des *-Pfades. Findet man einen Pfad der Länge 2, z.B. (A, B) nicht, sucht man erst in $(A, *)$ und (falls nicht gefunden) in */* und benutzt dessen Selektivität. Wenn alle zur Selektivitätsberechnung benutzten Pfade nur aus * bzw. */* Pfaden bestehen, wird die Selektivität mit Null angenommen.

4 XPathLearner

4.1 Einführung

Die Abschätzung der Selektivität eines XML Pfadausdrucks basiert auf zusammenfassenden Statistiken über die Struktur der zu Grunde liegenden Daten. Alle bisherigen Methoden sind darauf angewiesen, diese Daten einmal komplett einzulesen um die erforderlichen Statistiken erstellen zu können. Solche Off-line Scans sind aber in Anwendungen für das Internet oftmals nicht zulässig, da entweder die erforderlichen Daten nicht zugänglich sind oder zu gross sind um in ihrer Ganzheit gelesen zu werden. Desweiteren ignorieren alle bisherigen Methoden die Verteilung der Nutzlast und verschwenden damit einen grossen Teil des begrenzten Speicherplatzes für die Erstellung von Statistiken über unregelmässig abgefragte Teile der Daten. Idealerweise sollte mehr Speicherplatz für Statistiken über häufig abgefragte Teile der Daten verwendet werden. Ein weiterer Nachteil der vorgehenden Ansätze ist das Fehlen der Möglichkeit zur Aktualisierung der Statistiken. Dies ist für Internetanwendungen aber essentiell, da die zu Grunde liegenden Daten häufig einer andauernden Änderung unterliegen. Periodische Off-line Scans sind in diesem Fall weder effizient noch effektiv, da die Kosten für das Lesen der Daten mit ihrer Grösse steigt. Als letzte Einschränkung der bisherigen Methoden ist noch die Beschränkung auf einfache Pfadausdrücke (siehe 4.2) zu erwähnen.

XPathLearner ist ein System, das versucht, die dargestellten Schwächen der bisherigen Methoden zu eliminieren. Es liefert eine Abschätzung für die Selektivität eines Pfadausdrucks ohne die zu Grunde liegenden Daten eingelesen zu haben. XPathLearner sammelt die benötigten Statistiken über die Verteilung der Daten ausschließlich über einen Feedbackmechanismus, welcher es dem System gestattet sich der Nutzlast anzupassen und auf Veränderungen in den Daten zu reagieren. Damit wird mehr Speicherplatz für häufig benötigte Statistiken verwendet und man umgeht das erneute Einlesen der kompletten Daten bei Änderungen. Desweiteren ist XPathLearner nicht auf einfache Pfadausdrücke beschränkt, sondern erweitert die Selektivitätsabschätzung auf Pfade, die mit einem Datenwert enden, sowie auf Pfade, die mehrere Datenwerte enthalten.

4.2 Begriffe und Problemstellung

Ein XML Dokument ist von der Struktur her ein Baum, in dem jeder Knoten mit einem Bezeichner oder einem Wert assoziiert ist. In der Praxis sind Werte fast immer Blätter. Ein XML *Datenbaum* ist ein Baum, der aus mehreren Dokumenten entweder durch Zusammenfügen der Wurzeln der einzelnen Dokumente sofern diese gleich sind, oder durch Einführen eines neuen, gemeinsamen Wurzelknotens konstruiert wurde.

Ein *einfacher Pfadausdruck* p der Länge $|p|$ ist eine Folge von Bezeichnern $\langle t_1, t_2, \ldots, t_{|p|} \rangle, t_i \in \Sigma$, wobei Σ die Menge aller möglichen Bezeichner ist. Die Abfolge der Bezeichner in einem Pfadausdruck gibt einen Navigationspfad durch den XML Datenbaum an, wobei jedes Paar (t_i, t_{i+1}) der Abfolge einer gerichteten Kante im XML Datenbaum mit den Bezeichnern (t_i, t_{i+1}) entspricht. In XPath Notation kann ein solcher Navigationspfad auch als $//t_1/t_2/\ldots/t_{|p|}$ geschrieben werden.

Ein *mehrwertiger Pfadausdruck* ist ein einfacher Pfadausdruck, bei dem Werte mit einem oder mehreren Bezeichnern im Pfadausdruck assoziiert sind. Ein Spezialfall des mehrwertigen Pfadausdrucks bildet der *einwertige Pfadausdruck* bei dem nur der letzte Bezeichner der Folge mit einem Wert assoziiert ist.

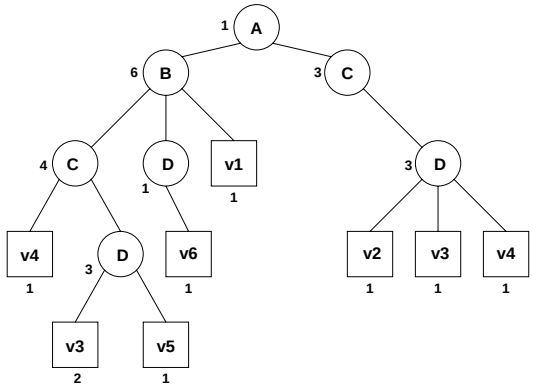


Abbildung 11: Beispiel eines Path Trees, vorgestellt in Abschnitt 2

Abbildung 11 zeigt einen Path Tree. Der Pfad $//B/C/D$ ist ein einfacher Pfadausdruck, der Pfad $//B/C/D = v3$ beschreibt einen einwertigen Pfadausdruck und $//B/C = v4/D = v3$ gibt einen mehrwertigen Pfadausdruck an.

Im Folgenden bezeichnet $\sigma(p)$ die Selektivität eines (einfachen, ein- oder mehrwertigen) Pfadausdrucks p. Die Selektivität eines einfachen Pfadausdrucks p ist definiert als die Anzahl der Pfade im XML Datenbaum, die der Bezeichnerfolge p entsprechen. Die Selektivität eines einwertigen Pfadausdruck ist ähnlich zu der eines einfachen Pfadausdrucks, außer dass der Navigationspfad mit einem Wert anstatt eines Bezeichners enden muss. Die Selektivität eines mehrwertigen Pfadausdrucks p ist definiert als die Anzahl der Teilbäume, die der Bezeichner- und Wertefolge in p entsprechen.

Als *Anfragefeedback* bezeichnet man das Tupel $(p, \sigma(p))$ bestehend aus dem Pfadausdruck und seiner entsprechenden realen Selektivität.

Problemstellung: Es soll die Selektivität eines gegebenen Pfadausdrucks (einfach, ein- oder mehrwertig) abgeschätzt werden, unter der Annahme, dass die notwendigen Statistiken nur über Anfragefeedback gewonnen werden können und dass nur ein begrenzter Speicherplatz für diese Statistiken zur Verfügung steht.

4.3 Erlernen der Pfadselektivitäten

Um das oben gestellte Problem zu lösen ist es zunächst notwendig, geeignete Datenstrukturen für das Speichern der Statistiken zu finden und Algorithmen zu entwerfen um diese Darstellung zu aktualisieren und zu verfeinern. In den folgenden Unterabschnitten wird zunächst ein Überblick über das System gegeben und dann auf die genaue Umsetzung eingegangen.

4.3.1 Überblick

Der Informationsfluss in XPathLearner ist in Abbildung 12 dargestellt. Das System lässt sich im Wesentlichen in drei Komponenten unterteilen:

- Histogramm Learner
- Markov Histogramm
- Selektivitätsabschätzer

Die Komponente 'Histogramm Learner' ist für das Erlernen der Statistiken zuständig, welche dann in der Komponente 'Markov Histogramm' gespeichert werden. Der 'Selektivitätsabschätzer' berechnet dann auf Basis dieser Statistiken die geschätzte Selektivität für eine gegebene Anfrage.

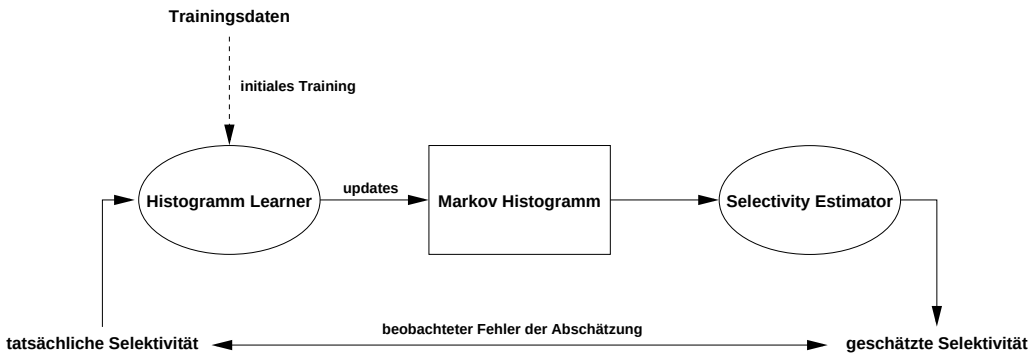


Abbildung 12: Informationsfluss in XPathLearner

Für den Lernprozess wird davon ausgegangen, dass das System für jede bearbeitete Anfrage eine Rückmeldung in Form der tatsächlichen Selektivität erhält. Diese Information wird von der Komponente 'Histogramm Learner' verwendet, um die Statistiken für den entsprechenden Pfad zu erlernen bzw. zu verfeinern. Dies geschieht auf Basis des beobachteten Fehlers zwischen geschätzter und tatsächlicher Selektivität unter Verwendung bestimmter Regeln zur Verteilung der Feedbacks auf die einzelnen Komponenten des Pfades. Die Komponente 'Histogramm Learner' kann ausserdem durch zusätzliche Trainingsdaten Statistiken erlernen. Dies setzt jedoch ein Wissen über den Aufbau der angefragten Daten voraus und ist in Internet Applikationen nicht immer gegeben.

4.3.2 Darstellung

XPathLearner speichert die Statistiken in Markov Histogrammen erster Ordnung. Ein Markov Histogramm der Ordnung $(m-1)$ ist eine Tabelle, die eine Menge unterschiedlicher Pfade bis zur Länge m zusammen mit der entsprechenden Selektivität enthält (siehe Abschnitt 3). Ein Markov Histogramm der Ordnung $(m-1)$ beruht auf der Annahme, dass das Auftreten eines bestimmten Bezeichners nur von den $(m-1)$ vorhergehenden Bezeichnern abhängt(siehe Seite 7). Diese Annahme ist realistisch für reale XML Dokumente für $m = 2$ und $m = 3$.

Selektivitätsabschätzung Wie in Abschnitt 3 beschrieben, kann die Selektivität eines einfachen Pfadausdrucks unter Benutzung eines Markov Histogramms erster Ordnung durch Gleichung (2) abgeschätzt werden.

Die Selektivität eines einwertigen Pfadausdrucks $p = //t_1/t_2/\ldots/t_{n-1} = v_{n-1}$ berechnet sich wie die Selektivität eines einfachen Pfadausdrucks, ausser dass in Gleichung (2) der Term $f(t_{n-1}, t_n)$ durch $f(t_{n-1}, v_{n-1})$ ersetzt wird. Die Selektivität eines mehrwertigen Pfadausdrucks $p = //t_1 = v_1/t_2 = v_2/\ldots/t_n = v_n$ kann durch

$$\begin{aligned} \hat{\sigma} \quad & (//t_1 = v_1/t_2 = v_2/\ldots/t_n = v_n) \\ = \quad & \hat{\sigma}(//t_1/\ldots/t_n = v_n) \cdot \left(\prod_{i=1}^{n-1} \frac{f(t_i, v_i)}{\sum_v f(t_i, v)} \right) \end{aligned}$$

berechnet werden. Dabei gibt $\frac{f(t_i,v_i)}{\sum_v f(t_i,v)}$ die Wahrscheinlichkeit für das Auftreten des Wertes v_i nach dem Bezeichners t_i an.

Datenwerte Die Benutzung von Markov Histogrammen erster Ordnung erfordert es, alle Pfade bis zur Länge 2 zu speichern. Diese Pfade beinhalten die Kombinationen Bezeichner-Bezeichner und Bezeichner-Wert. Diese beiden Kombinationen müssen bei der Speicherung unterschiedlich behandelt werden, da die Menge der Werte in einem XML-Dokument sehr viel grösser als die Menge der Bezeichner ist.

Bez.	$f(t_i)$
A	1
B	6
C	7
D	7

Bez.	Bez.	$f(t_i, t_{i+1})$
A	B	6
A	C	3
B	C	4
B	D	1
C	D	6

Bez.	Wert	$f(t_i, v_i)$
D	v3	3

Bez.	Merkmal	$f(t_i, v)$	Paare
B	a	1	1
C	a	1	1
D	a	2	2
D	b	2	2

Abbildung 13: Markov Histogramm zur Abbildung 11 unter der Annahme, dass die Datenwerte {v1,v2,v3,v4} das Merkmal 'a' und die Datenwerte v5,v6 das Merkmal 'b' besitzen.

Wenn ausreichend Speicherplatz zur Verfügung steht werden die Bezeichner-Bezeichner-Paare exakt abgespeichert. Andernfalls können Techniken wie in Abschnitt 3.1 beschrieben zum Einsatz kommen. Für die Bezeichner-Werte-Paare verwendet man folgende Strategie:

1. Die ersten k Einträge mit den grössten Häufigkeiten werden exakt abgespeichert, dabei ist k ein veränderlicher Parameter. Jeder Eintrag in der Datenstruktur hat die Form $\langle Bezeichner, Wert, Anzahl \rangle$.

2. Bezeichner-Werte-Paare deren Auftreten nicht unter die k häufigsten fallen werden in Containern zusammengefasst. Ein Container ist dabei ein Tupel der Form $\langle Merkmal,\ Häufigkeit,\ Paare \rangle$. Das Feld $Paare$ gibt die Anzahl der Bezeichner-Werte-Paare an, die in diesem Container zusammengefasst sind, das Feld $Häufigkeit$ enthält die Summe der Auftreten der einzelnen Paare und das Feld $Merkmal$ enthält das Merkmal des entsprechenden Containers. Die Zuweisung eines Bezeichner-Werte-Paar an einen Container erfolgt nach dem Merkmal des Paares.

Das Merkmal sollte so gewählt sein, dass die Häufigkeiten der Paare in einem Container möglichst gleich sind, d.h. die Varianz der Häufigkeiten der Paare in einem Container sollte minimiert werden.

Beispiel Abbildung 13 stellt das zum Path Tree aus Abbildung 11 gehörige Markov Histogramm erster Ordnung dar. Es wurde k=1 gewählt und angenommen, dass die Datenwerte {v1,v2,v3,v4} das Merkmal 'a' und die Datenwerte {v5,v6} das Merkmal 'b' besitzen.

Die Selektivität des einfachen Pfadausdrucks $//B/C/D$ kann unter Verwendung der Markov Histogramme aus Abbildung 13 und der Formel (2) durch

$$\hat{\sigma}(//B/C/D) = \frac{f(BC)}{f(C)} \cdot f(CD) = \frac{4}{7} \cdot 6 = 3.43$$

abgeschätzt werden. Analog verfährt man bei der Abschätzung der Selektivitäten von ein– und mehrwertigen Pfadausdrücken.

4.3.3 Update-Algorithmen

XPathLearner kennt zwei unterschiedliche Methoden zur Aktualisierung der Statistiken der Markov Histogramme: die Heavy-Tail-Regel und die Delta-Regel.

Die Algorithmen entsprechen dem Pseudocode in Abbildung 15 und unterscheiden sich nur in den verwendeten Aktualisierungsstrategien in Zeile 7. Das verwendete Markov Histogramm ist zu Beginn leer. Die Funktion `compress-add entry` fügt einen beliebigen unbekannten Pfad der Länge 2 zum Markov Histogramm hinzu. Falls der Speicherplatz zu knapp wird, ruft die Funktion Algorithmen zur Komprimierung des Histogramms auf (siehe Abschnitt 3.1). Im Gegensatz zum Erlernen der Bezeichner und Werte steht das Erlernen der entsprechenden Selektivität. Dies geschieht in Zeile 7 des Algorithmus.

Der Algorithmus lässt sich grob in drei Teile untergliedern:

1. Zeile 1 bis 3: Erlernen der Pfade bis zu einer Länge von 2

Bez.	$f(t_i)$
A	1
B	6
C	7
D	10
E	4

Bez.	Bez.	$f(t_i, t_{i+1})$
A	B	6
A	C	3
A	D	3
B	C	4
B	D	1
C	D	6
D	E	4

Bez.	Wert	$f(t_i, v_i)$
D	v3	3

Bez.	Merkmal	$f(t_i, v)$	Paare
B	a	1	1
C	a	1	1
D	a	2	2
D	b	2	2

Abbildung 14: Resultierendes Markov Histogramm nach einem Durchlauf des Update-Algorithmus

2. Zeile 4 bis 8: Erlernen längerer Pfade durch Zerlegung des Pfades in Pfade der Länge 2
3. Zeile 9 bis 12: Erlernen von einzelnen Tags

Der Prozess des Erlernens ist immer mit einer Aktualisierung schon vorhandener Statistiken verbunden. Diese Aktualisierungen können auf zwei verschiedene Arten durchgeführt werden. Sie werden in den beiden folgenden Abschnitten beschrieben.

Beispiel Das Markov Histogramm sei im Zustand aus Abbildung (13). Die geschätzte Selektivität des Pfadausdrucks $p = //A/D/E$ ist 1, da dieser Pfad nicht im Histogramm vorkommt, als Feedback liege eine Selektivität von $(p, 5)$ vor. Der Update-Algorithmus nimmt nun folgende Aktualisierungen vor:

- Zunächst wird überprüft, ob es sich um einen Pfad der Länge 2 oder kleiner handelt (Zeile 1 bis 3). Dies ist nicht der Fall.
- Nun wird der Pfad in Pfade der Länge 2 zerteilt. Für jeden dieser Teilpfade wird überprüft, ob ein entsprechender Eintrag im Markov Histogramm existiert. In unserem Beispiel handelt es sich um die Pfade $/A/D$ und $/D/E$. Beide kommen nicht vor, werden also mit einer Selektivität von 1 hinzugefügt.
- Als nächstes wird der Fehler der Abschätzung auf die einzelnen Unterpfade verteilt (Zeile 7). Nimmt man hier als Updatestrategie die Heavy-Tail-Regel (siehe Abschnitt 4.3.3) an, so ergeben sich folgende Updates für die Teilpfade der Länge 2:

$$\begin{aligned} f(A/D) &= round(1 + 4^{1/3}) \approx 3 \\ f(D/E) &= round(1 + 4^{2/3}) \approx 4 \end{aligned}$$

- Als letztes erfolgen die Updates für die einzelnen Bezeichner (Zeile 9 bis 12). In diesem Fall sind dies

$$\begin{aligned} f(D) &= \max\{7, 1 + 6 + 3\} = 10 \\ f(E) &= \max\{1, 4\} = 4 \end{aligned}$$

Das resultierende Markov Histogramm ist in Abbildung (14) dargestellt.

Die Heavy-Tail-Regel Kennt man die geschätzte Selektivität $\hat{\sigma}(p)$ eines Pfades $p = //t_1/\ldots/t_n$ und hat das entsprechende Queryfeedback $(p, \sigma(p))$ zur Verfügung, so kann man den beobachteten Fehler

$$\epsilon = \sigma(p) - \hat{\sigma}(p) \tag{3}$$

berechnen. Die Selektivität des Pfades p berechnet sich nach (2) als

$$\hat{\sigma}(//t_1/\ldots/t_n) = \left(\prod_{i=1}^{n-2} \frac{f(t_i, t_{i+1})}{f(t_{i+1})} \right) \cdot f(t_{n-1}, t_n).$$

```
UPDATE(Mhistogram f, Feedback (p,σ), Estimate σ̂)
 1  if |p| ≤ 2 then
 2    if not exists f(p)
        then compress-add entry f(p) = σ
 3    else f(p) ← σ
 4  else
 5    for each (t_i, t_{i+1}) ∈ p
 6      if not exists f(t_i, t_{i+1})
           then compress-add entry f(t_i, t_{i+1}) = 1
 7      f(t_i, t_{i+1}) ← update
 8    endfor
 9  for each t_i ∈ p, i ≠ 1
10    if not exists f(t_i)
        then compress-add entry f(t_i)
11    f(t_i) ← max f(t_i), Σ_α f(α, t_i)
12  endfor
```

Abbildung 15: Pseudocode zur Aktualisierung des Histogramms durch Anfragefeedback. '←' bezeichnet eine Aktualisierung eines vorhandenen Eintrags

Alle Terme $f(t_i, t_{i+1})$ in diesem Produkt müssen nun auf Basis des beobachteten Fehlers ϵ verfeinert werden.

Weiterhin ist es sinnvoll den Termen am Ende eines Pfades einen grösseren Anteil des beobachteten Fehlers anzulasten. Dies hat zwei Gründe: erstens sind die Terme am Ende eines Pfades in der Regel relevanter für die Selektivitätsabschätzung und zweitens möchte man den Effekt eines Updates auf andere Pfade, die den gleichen Präfix haben, minimieren. Deshalb gewichtet man die Terme $f(t_i, t_{i+1})$, wobei das Gewicht mit steigendem i zunimmt.

Sei nun w_i das Gewicht, das mit t_i assoziiert ist. Die Aktualisierung des Terms $f(t_i, t_{i+1})$ erfolgt nun nach der Formel

$$f_{l+1}(t_i, t_{i+1}) \leftarrow f_l(t_i, t_{i+1}) + sign(\epsilon)(\gamma|\epsilon|)^{w_i / \sum_{j<n} w_j}, \tag{4}$$

wobei $\sum_{j<n} w_j$ der Normalisierungsfaktor für die Gewichte ist, γ die Lernrate beschreibt, t_i der i-te Bezeichner im Pfad p ist, und $f_l(\cdot)$ und $f_{l+1}(\cdot)$ die Häufigkeiten vor und nach der Aktualisierung der Terme angeben.

Die Lernrate sollte kleiner als 1 gewählt werden, so dass die Fehlerberichtigung verringert wird. Dies verhindert eine Überreaktion auf den Abschätzungsfehler und glättet den Prozess der Fehlerreduktion. Die Terme $f(t_i)$ werden durch

$$f_{l+1}(t_i) \leftarrow \max\{\sum_j f_{l+1}(t_j, t_i), f_l(t_i)\} \tag{5}$$

aktualisiert, da die Summe aller Pfade der Länge 2 die auf t_i enden nach

$$f(t_i) \geq \sum_{\alpha \in Bezeichner/Wert} f(\alpha, t_i)$$

eine untere Schranke für $f(t_i)$ bilden. Der Term $f_l(t_i)$ kann trotzdem grösser als $\sum_j f_{l+1}(t_j, t_i)$ sein, falls vorher bereits ein Feedback für einen Pfad t_i der Länge 1 vorlag.

Als Beispiel nehmen wir an, dass das Markov Histogramm sich in dem in Abbildung 13 angegebenen Zustand befindet. Weiterhin sei das Feedback für den Pfad ACD $(ACD, 6)$ und seine geschätzte Selektivität sei $\hat{\sigma} = 3 \cdot 6/t \approx 3$. Der beobachtete Fehler ist dann $\epsilon = 6 - 3 = 3$. Mit $\gamma = 1$ werden folgende Updates vorgenommen:

$$f_{l+1}(AC) \quad \leftarrow \quad round(3 + 3^{1/3}) = 4,$$

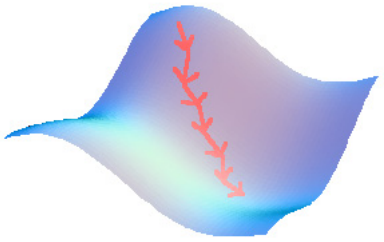


Abbildung 16: Visualisierung der Gradientenmethode

$$\begin{aligned} f_{l+1}(CD) &\leftarrow round(6+3^{2/3}) = 8, \\ f_{l+1}(C) &\leftarrow \max\{4+4,7\} = 8, \\ f_{l+1}(D) &\leftarrow \max\{1+8,7\} = 9. \end{aligned}$$

Die geschätzte Selektivität nach der Aktualisierung des Pfades ACD ist dann $4 \cdot 8/8 = 4$. Der Fehler in der Abschätzung wurde also um 1 reduziert.

Die Delta-Regel Die Delta-Regel ist eine fehlerreduzierende Lerntechnik, welche oft auch in Gebieten der KI zum Einsatz kommt. Sie basiert auf der Methode des steilsten Abstiegs, der Gradientenmethode. Diese Methode versucht ein globales Minimum einer Funktion zu bestimmen, indem sie, graphisch gesehen, sich in die Richtung der grössten Änderung bewegt. Die Richtung der grössten Änderung einer Funktion oder eines Vektorfeldes ist durch den Gradienten der Funktion oder des Vektorfeldes bezüglich eines Parameters gegeben. Mathematisch ausgedrückt entspricht dies einer partiellen Ableitung der Funktion nach dem Parameter. Stellt man dies graphisch dar, so ergibt sich Abbildung (16). Man erkennt, dass die Schrittweite des Abstieges entscheidend für die Konvergenz bzw. das Auffinden des Minimums ist. Ist sie zu klein gewählt, d.h. sind die Änderungen zu klein, so konvergiert die Methode nur sehr langsam. Ist sie auf der anderen Seite zu gross gewählt, kann das globale Minimum überschritten werden.

Die Eingaben seien wieder die gleichen wie bei der Heavy-Tail-Regel. Der Histogramm Learner erhält die geschätzte Selektivität $\hat{\sigma}(p)$ und das Queryfeedback $(p, \sigma(p))$. Der beobachtet Fehler ϵ berechnet sich wie in (3).

Die Delta-Regel minimiert eine Fehlerfunktion. Wir wählen hier den quadrierten Fehler

$$E = (\sigma(p) - \hat{\sigma}(p))^2. \tag{6}$$

Die Delta-Regel besagt, dass das Update zum Term $f(t_i, t_{i+1})$ proportional zum negativen Gradienten von E bezüglich $f(t_i, t_{i+1})$ sein sollte, mathematisch ausgedrückt

$$f(t_i, t_{i+1})^{(l+1)} \leftarrow f(t_i, t_{i+1})^{(l)} - \gamma \frac{\partial E}{\partial f(t_i, t_{i+1})}, \tag{7}$$

wobei γ wieder die Lernrate oder Schrittweite angibt.

Diese Gleichung lässt sich unter Verwendung von Gleichung (2) und der Quotientenregel für die Differentiation umformen, so dass man eine Vorschrift für die Aktualisierung der Terme bekommt, die nur Einträge des Histogramms benutzt. Eine Ausführung dieser Vereinfachung trägt aber nicht wesentlich zum Verständnis der Methode bei und unterbleibt daher an dieser Stelle.

5 XSketch

5.1 Einführung

In diesem Teil entwickeln und benutzen wir eine statische Zusammenfassung eines großen XML Datengraphs. Das vorgeschlagene Zusammenfassungsmodell (XSketch-Zusammenfassung) lokalisiert die Graphsstabilität, um in beschränktem Speicherraum die Pfadverteilung im Datengraphen genau wiederzugeben. Wir entwickeln einen Abschätzungsmodell, das sich statistischer Annahmen (Gleichförmigkeit und Unabhängigkeit) bedient, um den Mangel der exakten Informationsverteilung zu kompensieren. Das Problem der Konstruktion eines optimalen XSketch für ein gegebenes Speicherbudget ist NP-vollständig. Um eine effiziente Zusammenfassung des XML Datengraphen zu gewährleisten, benötigt man Verfeinerungsoperationen. Der Aufbau des XSketches durch aufeinanderfolgende Verfeinerungen des Graphs wird lokal gemacht und versucht interessante statistische Korrelation zwischen den Elementen des Datengraphs zu erhalten. Die experimentellen Ergebnisse, die die Effizient von XSketch-Zusammenfassungen nachprüfen, haben gezeigt, dass XSketches besser als frühere Techniken wie zum Beispiel Markov Tabellen sind.

5.2 XSketchzusammenfassung

Wir geben eine kurze Erklärung des Problems des Konstruierens und Verwendens solcher Graphzusammenfassungen ab, um die Grundlagen für die Abschätzung des Pfadausdrucks zu legen.

5.2.1 Allgemeines Graphzusammenfassungsmodell

Das allgemeine Modell der Zusammenfassung für einen XML Datengraphen $G = (V_G, E_G)$ ist ein gerichteter, strukturierter Graph $S(G) = (G_S, E_S)$, dessen Knoten ausgezeichnet sind. In Abbildung (17) ist eine solche Zusammenfassung dargestellt. Dabei gilt

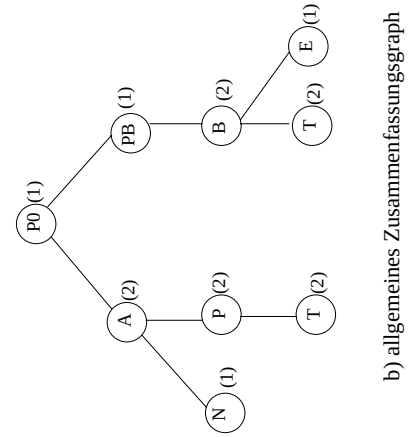
(a) Jeder Knoten $v \in (G_S)$ entspricht einer Teilmenge von identisch bezeichneten Knoten in einer Partition von (V_G), welche Ausdehnung (engl. extent) von v heißt. Wir bezeichnen diese Menge im Folgenden mit *extent(v)*. Ein solcher Knoten enthält zusammenfassende Information über G in der Form eines Zähler-Feldes (*count(v)*), der die Kardinalität von *extent(v)* aufnimmt (Größe der Ausdehnung von v).

(b) Eine Kante $(u, v) \in E_S$ exisistiert genau dann, wenn Knoten in *extent(u)* und *extent(v)* existieren, so dass eine Kante zwischen diesen Knoten im ursprünglichen Graphen existierte.

5.2.2 Definition der XSketch-Zusammenfassungs

XSketch Synopsestrukturen stellen eine Instanz des allgemeinen Graphzusammenfassungmodells dar. Wir präsentieren zwei Schlüsselkonzepte des zu Grunde liegendes XSketches: die Rückwärts– und Vorwärtsstabilität, "Backward-stability" und "Forward-stability".

Definitionen Seien V und U zwei Mengen von Elemente in einem XML Daten Graph G. Rückwärtsstabilität: V ist rückwärtstabil(B-stabil) in Bezug auf U genau dann, wenn für jedes $v \in V$, ein Element $u \in U$ exisistiert, so dass die Kante $(u, v) \in G$ ist. Vorwätsstabilität: U ist vorwärtsstabil (F-stabil) in Bezug auf V genau dann, wenn für jeden Knoten $u \in U$ ein Knoten $v \in V$ existiert, so dass die Kante $(u, v) \in G$ ist. Als Beispiel betrachten wir die Abbildung (18). Wir beobachten, dass alle Elemente in *extent(P)* ein Vaterelement in *extent(A)* haben. Deshalb ist der Knoten P B-stabil in Bezug auf A. Das Zählfeld assoziiert mit Knoten P ergibt offensichtlich eine genaue Abschätzung für die Anzahl von Elementen, die durch den Pfadausdruck A/P erreicht werden. Sei $S(G)$ eine Graphzusammenfassung eines Graphen G. Wir definieren einen Knoten w in der Zusammenfassung eines Graphs G, als B-stabil (F-stabil) in Bezug auf anderen Knoten x genau dann, wenn die Ausdehnung von W *extent*(w) B-stabil (resp. F-stabil) in Bezug auf *extent*(x) ist.



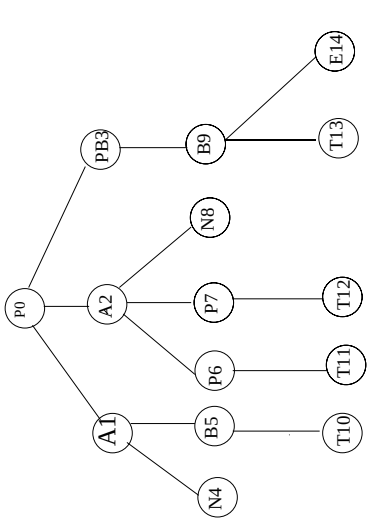


Abbildung1

Abbildung 17: Datengraph und zugehörige Zusammenfassung

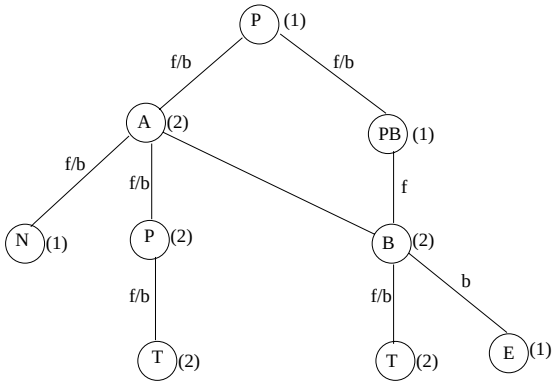
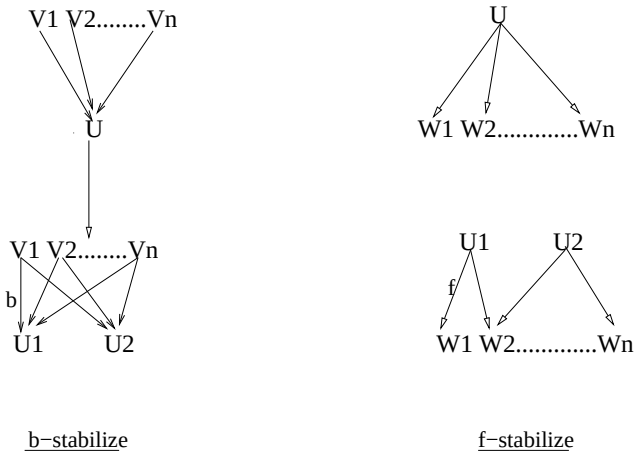


Abbildung2. XSketchzusammenfassungsgraph

Abbildung 18: XSketch-Zusammenfassung



b−stabilize

f−stabilize

Abbildung3

Abbildung 19: Operationen b-stabilize und f-stabilize

Ein XSketch $X_S(G) = (VX_S, EX_S)$ für XML Datengraphen G ist eine kantenbeschriftete Graphzusammenfassung für G, wobei der Bezeichner für jede Kante $(u, v) \in EX_S$ ein 2-Bit-Indikator ist, dessen Wert wie folgt definiert wird:

1. *Label(u, v)* = *{b}*, wenn v B-stabil in Bezug auf u ist,
2. *Label(u, v)* = *{f}*, wenn u F-stabil in Bezug auf v ist,
3. *Label (u, v)* = *{b, f}*, wenn die(1) und (2) gilt,
4. *Label (u, v)* = $\emptyset$, sonst.

Das Beispiel in Abbildung (19) zeigt die XSketch-Zusammenfassung eines XML Datengraph.

5.3 Abschätzungsmodell

Um die Abschätzung eines komplizierten Pfadausdruckes über einem XSketch $Xs(G)$, der auf einem XML Datengraphen G basiert, durchzuführen, definieren wir ein Abschätzungsmodell (engl.

estimation framework). Sei *L=l1/../ln* ein einfacher Pfadausdruck über einen XML Datengraphen G. Wir betrachten $Xs(G)$ von G und einen Pfadausdruck *V=V1/../Vn* mit *Label(Vi) = li* für alle *i*. Die Abschätzung der Selektivität des Pfadausdrucks *V* ist *count(V1/./Vn)* mit: *count(V1/./Vn) = count(Vn)*f(V1/./Vn)*. Dabei ist *f(V1./Vn)* die Wahrscheinlichkeit für die Elemente in *extent(Vn)*, die durch den Pfadausdruck *V1/../Vn* erreichbar sind. Um diese Wahrscheinlichkeit zu bestimmen, haben wir zwei Möglichkeiten: Falls alle Kanten die Vorwärtsstabilität erfüllen, dann ist *f(V1/./Vn)* = 1. Daraus folgt, dass *count(V1/../Vn) = count(Vn)*, d.h. die Selektivität des Pfadausdrucks *V1/../Vn* ist gleich der Anzahl der Knoten in *extent(Vn)*, die man leicht berechnen kann.

Die Kanten *(A,P)* und *(P,T)* aus dem Graph in der Abbildung (18) erfüllen die Vorwätsstabilität. Daraus folgt, dass *f(A/P/T)* = 1. Die Selektivität des Pfadausdrucks *A/P/T* ist *count(A/P/T) =count (T)*f (A/P/T)*= 2. Der andere Fall tritt auf, wenn eine Kante *(Vi,Vi+1)* in dem einfachen Pfadausdruck *V1/./Vn* die Vorwätsstabilität nicht erfüllt. Wir zerlegen diesen Pfadausdruck in zwei Teile *V1/./Vi* und *Vi+1/../Vn*. Zur Bestimmung der Selektivität dieses Pfadausdrucks brauchen wir zwei Annahmen.

A1–Pfadunabhängigkeit Gegeben ist ein Knoten $v \in Xs(G)$, Die Verteilung der eingehenden Pfade zu v ist unabhängig von der Verteilung der ausgehenden Pfade von v d.h. $f(u/v|w/v) \approx f(u/v)$. $f(u/v|w/v)$ bezeichnet die Wahrscheinlichkeit, dass das Datenelement in einer Zusammenfassung von Knoten v vom Pfadausdruck u/v entdeckt wird unter der Bedingung, dass der Pfadausdruck v/w existiert.

A2–Gleichförmigkeit : Wir betrachten einen Knoten $v \in Xs(G)$. Die eingehenden Kanten zu v von allen Vaterknoten u , die nicht die Eigenschaft der B-Stabilität erfüllen, sind gleichförmig über die Menge aller solcher Knoten im Verhältnis zu ihrem Auftreten verteilt. Das folgende Beispiel zeigt die Anwendung der beiden Annahmen zur Berechnung der Selektivität des Pfadausdrucks *P/PB/B/E* aus dem Graph in Abbildung (18).

*f(P/PB/B/T) = f(B/T)*f(P/PB/B * B/T)*
*= f(B/T)*f(PB/B * B/T)*f(P/PB * PB/B/T)*
*= f(PB/B * B/T)*
= f(PB/B)
= count(PB)/[count(PB) + count(A)]
= 1 / 1+2 = 1/3.

Zur Abschätzung eines komplexen Pfadausdrucks benötigen wir zwei weitere Annahmen.

A3–Unabhängigkeitsannahme : Gegeben ist ein Knoten $v \in Xs(G)$. Pfadausdrücke aus dem Knoten v sind unabhängig von der Existenz anderer Pfadausdrücke v/w aus dem Knoten v.

A4–Gleichförmigkeit der Vorwärtsstabilität : Die Kanten aus v zu allen Kindern u, so dass v nicht F-stabil in Bezug auf u ist, werden gleichförmig über alle diese Kinder in Verhältnis zu ihrem Auftreten verteilt.

5.4 XSketch-Verfeinerungsoperationen

Um einen effizienten XSketch für ein gegebenes Speicherbudget zu bauen, müssen wir die Zusammenfassungs-Struktur für Gebiete des Datengraphen, in denen unsere Gleichförmigkeits– und Unabhängigkeitsannahme versagen, angemessen verfeinern, da diese Gebiete zu hohen Abschätzungsfehlern führen können. Um die Abschätzungsgenauigkeit zu verbessern, stellen wir die Verfeinerungsoperationnen vor, um das Stück der Zusammenfassung zu verfeinern, in dem eine der Abschätzungsannahmen (A1) - (A4) nicht auftritt. Wir definieren drei verschiedene Verfeinerungsoperationnen für XSketch-Knoten, nämlich b-stabilize, f-stabilize und b-slit.

b-stabilize(Xs(G), *u*, *v*): Sei v ein Vaterknoten des Knotens u in $Xs(G)$, so dass u nicht B-stabil in Bezug auf v ist. Die Operation b-stabilize verfeinert den Knoten u in zwei Partition

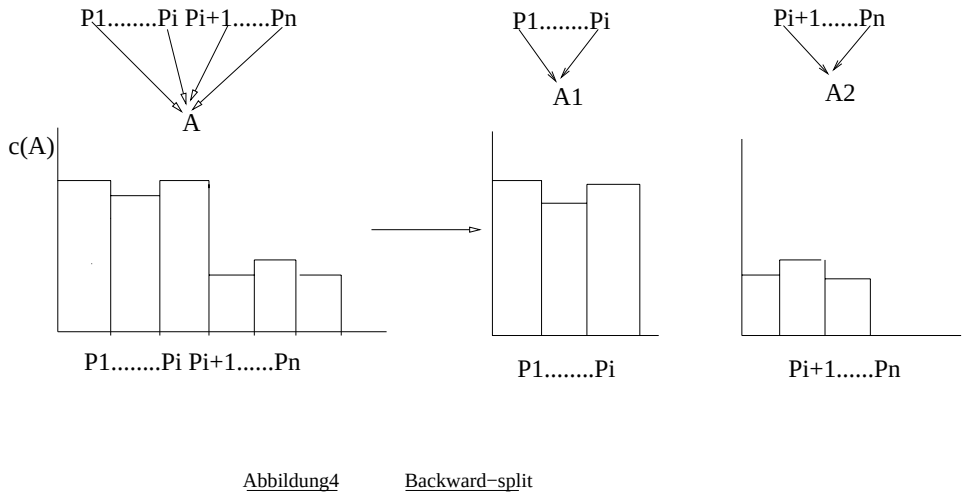


Abbildung 20: Operation b-split

(Abbildung (19)) mit gleichen Bezeichner, so dass eine der Partitionen B-stabil in Bezug auf v ist. Die b-stabilize-Operation trennt jedes Datenelement in u. *Label(v,u1)=Label(v,u)U{B}*

f-stabilize(Xs(G), u,w): Sei u ein Vaterknoten des Knotens w, so dass w nicht F-stabil in Bezug auf u ist. Die Operation f-stabilize trennt dieses Element von u in einen neuen Zusammenfassungsknoten *u1*, so dass *Label(u1, w) =Label(u, w) U F*(Abbildung (19)).

b-split(XS(G), u, vi): Sei u ein Knoten in $Xs(G)$ und vi eine Menge von Vaterknoten des Knotens u, so dass u B-stabil in Bezug auf vi ist. Die Operation b-split fasst die Elemente in u zu neuen XSketch-Knoten zusammen (Abbildung (20)), so dass

(1) Die Menge der Vaterknoten vi von Knoten u wird in zwei neue Knoten zerlegt,

(2) Die Annahme der Gleichförmigkeit der Vorwärtskanten liefert höhere Genauigkeit. Wie die Abbildung (20) zeigt: Die Menge der Vaterknotenvi wird durch clevere Zerlegung, die auf der Größe(count) der Informationen basiert, in gleichförmigere Mengen zerlegt. b-split erreicht eine gleichförmigere Größe des Histograms.

5.5 Experiment und Ergebnis:

In diesem Experiment vergleichen wir die XSketch-Zusammenfassung mit den Markov Tabellen. Wir betrachten einen Pfadausdruck der Größe P=1000 bzw. einen Knoten mit v=1% der gesamten Knoten in der Zusammenfassung. Abbildung (21) zeigt den Abschätzungsfehler der beiden Methoden als eine Funktion der Zusammenfassungsgröße über den DBLP–Daten. XSketches sind effizienter, da die Abschätzungen mit sehr niedrigem Fehler berechnet werden. Mit einem kleinen Raumbudget von 30 Kbytes hat XSketch einen Abschätzungsfehler von 6% gegenüber 19% für MT-Zusammenfassungen.

6 Zusammenfassung

Zusammenfassend kann gesagt werden, dass die Wahl der Methode von den zu Grunde liegenden Daten und dem Einsatzgebiet abhängt. In Anwendungen für das Internet hat XPathLearner als Online-System deutliche Vorteile gegenüber Offline-Systemen, da es in der Lage ist, auf Änderungen in den Daten zu reagieren. Allerdings ist dieses System auf baumstrukturierte Daten beschränkt. Sollen auch idref's und XLinks unterstützt werden, so sind XSketches die Methode der Wahl.

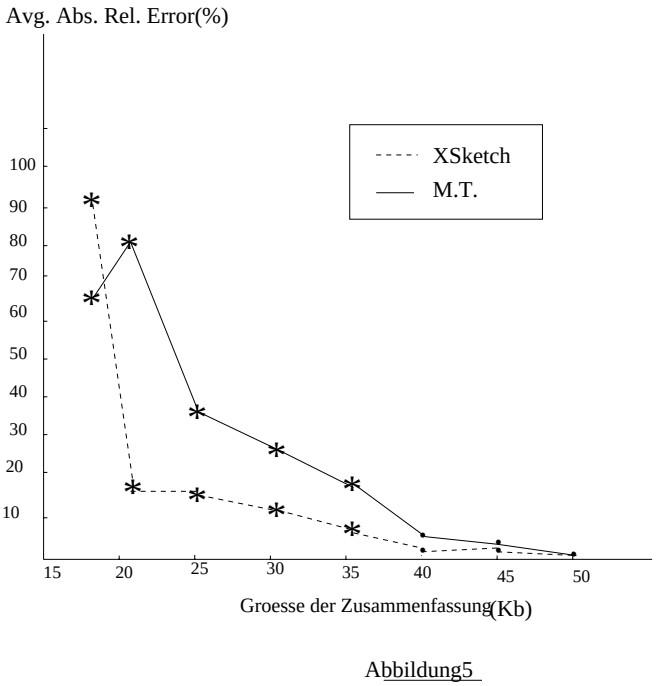


Abbildung 21: XSketch vs. Markov Tabellen

Alle Systeme müssen das Problem eines begrenzten Speichers behandeln. Bei allen Methoden bedeutet eine Zusammenfassung der Dokumentenstruktur einen Informationsverlust. Die Wahl der Kompressionsmethode sollte bei Markov Tabellen und Path Trees von den zu erwartenden Resultaten abhängig gemacht werden. Kann man davon ausgehen, dass der gesucht Pfadausdruck existiert, wählt man eine *-Methode, andernfalls eine No-*-Methode.

Der Grad der Zusammenfassung beinflusst, wie schon angedeutet, die Genauigkeit der Selektivitätsabschätzung. Nimmt man bei allen Methoden einen ähnlichen Kompressionsgrad an, so zeigen Experimente, dass XSketches einen geringen mittleren Fehler in der Abschätzung aufweisen als Markov Tabellen. Der Vergleich zwischen Online– und Offline-Systemen mit Markov Tabellen zeigt eine Ebenbürtigkeit des Onlinesystems nach einer gewissen Trainingsphase. Das Onlinesystem übertrifft die Offline-Methoden allerdings klar in Fragen der Anpassungsfähigkeit an Änderungen.

Literatur

[AAN] Ashraf Aboulnaga, Alaa R. Alameldeen, and Jeffrey F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications.

[Gar] Minos N. Garofalakis. Statistical Synopses for Graph-Structured XML Databases.

[LWP+] Lipyeow Lim, Min Wang, Sriram Padmanabhan, Jeffrey Scott Vitter, and Ronald Parr. XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation.

[Pfe00] Rene Pfeuffer. Approximative Anfragebearbeitung auf XML-Datenströmen. Master's thesis, Universität Passau, 2000.