Querybearbeitung mit Views
der MiniCon Algorithmus

Tim Jammer

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik, Arbeitsbereich Algorithmen, Randomisierung und Theorie
Seminar Informationsintegration

Inhaltsverzeichnis

1 Einleitung

Bei der Integration von Informationen aus verschiedenen Quellen ist es von essentieller Bedeutung, diese Informationen zu kombinieren, um eine globale Sicht auf die Informationen zu gewinnen. Dabei gibt es verschiedene Algorithmen, die im folgendem vorgestellt werden. Jede einzelne Informationsquelle wird durch einen eigenen View repräsentiert. Die Algorithmen berechnen dann zu einer gegebenen Benutzeranfrage auf dem globalen View, wie sich diese mithilfe der gegebenen Informationsquellen (Views) am besten beantworten lässt.
Abschnitt 2 führt dabei die formalen Definitionen für dieses Problem ein.
Diese werden in Abschnitt 3 anhand eines Beispiels erläutert. Auf Basis dieses Beispiels werden dann der Bucket Algorithmus (Abschnitt 4), der inverse-rule Algorithmus (Abschnitt 5) und der MiniCon Algorithmus (Abschnitt 6) vorgestellt.
Die Algorithmen werden anschließend in Abschnitt 7 miteinander verglichen, wobei sich herrausstellt, dass der MiniCon Algorithmus in allen Fällen bessere Performance bringt als die anderen vorgestellten Algorithmen. Das abschließende Fazit in Abschnitt 8 rundet die Arbeit ab.

2 Problemdefinition

Formal werden Queries und Views mithilfe von Prädikatenlogik definiert. Ein Query

$$q(X) := e_1(X_1), e_2(X_2), ..., e_i(X_i)$$

ist eine Formel der Prädikatenlogik. $q, e_1, e_2, ..., e_i$ sind dabei Prädikate. Sie drücken Relationen in der Datenbank aus. $q(X)$ ist dabei der Kopf des Queries. Das Prädikat q definiert die angefragte Zielrelation. Diese Relation ist durch die einzelnen Teilziele $e_1, e_2, ..., e_i$ definiert.
Die Tupel $X, X_1, X_2, ..., X_i$ sind Variabeln oder Konstanten. Es werden nur solche Queries betrachtet, in denen alle Variablen im Kopf des Queries auch in mindestens einem Teilziel vorkommen.[1] Variablen, die nur in den Teilzielen vorkommen, nicht aber im Kopf des Queries werden existentielle Variablen genannt, da die Anfrage nicht nach dieser Entität fragt, ihre Existenz in bezug auf die Teilziele aber fordert.

Die Bedeutung eines Queries ergibt sich aus Unifikation der Prädikatenlogik.
Ein Beispiel zur Veranschaulichung folgt im nächsten Abschnitt.

Views sind Queries, die einen Namen d.h. ein eindeutig definiertes Prädikat q haben. Das vom Benutzer angefragte Query wird in dieser Arbeit immer mit Q bezeichnet.

[1] Andernfalls kann man das Query nicht sinvoll beantworten, da man nicht weiß, welche Eigenschaften die nicht in den Teilzielen vorhandene Variable erfüllen soll.

Ein Query Q enthält ein Query Q', wenn bei jeder Datenbank-Instanz die Antwort auf Q' eine Teilmenge der Antwort auf Q sind. Man schreibt dann $Q' \sqsubseteq Q$. Q enthält Q' genau dann, wenn es eine Abbildung τ gibt, die jeder Variable in Q' auf eine in Q abgebildet wird, sodass jedem Teilziel aus Q' einem aus Q entspricht. [CM77]

Falls $Q' \sqsubseteq Q$ und $Q \sqsubseteq Q'$ gilt, so sind Q und Q' äquivalent. Dies ist gleichbedeutend damit, dass die Formeln $Q(X) \Rightarrow e_1(X_1), e_2(X_2), ..., e_i(X_i)$ und $Q'(X) \Rightarrow e_1(X_1), e_2(X_2), ..., e_i(X_i)$ im Sinne der Prädikatenlogik äquivalent sind.

Eine äquivalente Umformulierung eines Queries Q ist ein Query Q', bei dem als Teilziele nur die Views $V_1, V_2, .., V_i$ verwendet werden, wenn die Auswertung von Q und Q' auf jeder Datenbank-Instanz die gleichen Ergebnisse liefert. Also dann, wenn Q und Q' äquivalent sind, wenn man die Teilziele von Q' mit den entsprechenden Viewdefinitionen unifiziert. Ein Beispiel dazu ist im nächsten Abschnitt zu finden.

Im Kontext der Informationsintegration ist es oft nicht möglich eine äquivalente Umformulierung eines Queries zu finden. Daher sucht man nach einem Q maximal enthaltendem Query Q'. Man sucht also nach einem Query, sodass es kein anderes Query Q'' gibt, dass nur unter der Verwendung der gegebenen Views mehr Elemente der angefragten Relation Q beschreibt. Formal ist ein maximal enthaltenes Query Q' daher ein Query, derart dass $Q' \sqsubseteq Q$ gilt, sofern kein Query Q'' mit $Q' \sqsubseteq Q'' \sqsubseteq Q$ für $Q' \neq Q'' \neq Q$ gibt.

3 Beispiel

Der folgende Abschnitt erläutert die vorhergehenden Definitionen anhand eines Beispiels.

Im Beispiel seien die folgenden Relationen definiert:

- Flugzeug(x,y) Flugverbindung zwischen zwei Städten
- Bahn(x,y) Bahnverbindung zwischen zwei Städten
- Schiff(x,y) Schiffsverbindung zwischen zwei Städten
- amMeer(x) Die Stadt liegt am Meer
- gleichesLand(x,y) Die Städte liegen im gleichen Land

Ein Query ist dann beispielsweise Q(a):= Bahn(a,b),amMeer(b). Diese Anfrage beschreibt alle Städte, die eine Bahnverbindung zu einer Stadt am Meer haben. Anfragen mit Vergleichsoperatoren (wie $<, \leq, \neq$) werden in einer Erweiterung der vorgestellten Algorithmen behandelt [PH01], daher beschränken wir uns hier nur auf einfachere Anfragen.

Für das Beispiel seien die Views

- V_1(x,y)=gleichesLand(x,y),Bahn(x,y) Welcher alle Bahnverbindungen innerhalb der Länder beinhaltet.
- V_2(x)=amMeer(x) Welcher Alle Städte am Meer beinhaltet
- V_3(x,y)=Schiff(x,y),Flugzeug(x,y) Welcher alle Städte, die sowohl per Bahn als auch per Schiff verbunden sind, beinhaltet
- V_4(x)=Flugzeug(x,x),amMeer(x) Welcher alle Städte am Meer, von denen aus Rundflüge angeboten werden, beinhaltet

definiert.
Eine Umformulierung der Anfrage Q(a):= Bahn(a,b),amMeer(b) wäre dann $Q'(a) := V_1(a,b), V_2(b)$ Welches alle Städte beschreibt, die eine inländische Bahnverbindung zu einer Stadt am Meer haben. Es gilt $Q' \sqsubseteq Q$, Da Q alle Städte aus Q' enthält und zusätzlich noch die Städte, die nur eine internationale Bahnverbindung zu einer Stadt am Meer haben. Bei den gegebenen Views ist dies allerdings die maximal mögliche Neuformulierung des Queries, da es nicht möglich ist, aus den gegebenen Datenquellen, die Städte, die nur eine internationale Bahnverbindung zum Meer haben, zu erfahren. Auch das Query $Q_1 := V_1(a,b), V_4(b)$ ist in Q enthalten, allerdings enthält es weniger Städte, da nur nach Städten gefragt wird, in denen am Zielort (am Meer) Rundflüge angeboten werden. Q_1 ist daher nicht das maximale enthaltene Query.

4 Bucket Algorithmus

Die Idee des Bucket Algorithmus [RLJ96] ist es, den Suchraum der möglichen Kombinationen der Views einzuschränken, in dem man nur die Views betrachtet, die für mindestens ein Teilziel der Anfrage relevant sind.

Dazu wird zunächst in Phase 1 ein Bucket für jedes Teilziel der Anfrage Q angelegt. Jeder Bucket bekommt nun die Views zugewiesen, die für ein Teilziel von Q möglicherweise relevant sind, also diejenigen Views, bei denen es möglich ist, durch die Unifikation der Viewdefinition das Teilziel zu erfüllen.

Dazu ein Beispiel:
Seien die Viewdefinitionen wie in Abschnitt 3 gegeben. Für die Anfrage Q:=Q(a):= Bahn(a,b),amMeer(b) werden folgende Buckets gebildet:

- Bahn(a,b)
  - V_1
- amMeer(b)
  - V_2
  - V_4

In der zweiten Phase werden nun alle Queries die sich aus dem kartesischen Produkt der einzelnen Buckets bilden lassen daraufhin überprüft, ob sie in Q enthalten sind. Das Maximal in Q enthaltene Query Q' wird dann durch das Union aller dieser Queries gebildet. Im Beispiel sind dies die, die in Abschnitt 3

diskutiert wurden.

Schwäche dieses Algorithmus ist, dass die einzelnen Zeilen nur in Isolation betrachtet werden und man daher in Phase zwei möglicherweise viele unpassende Kombinationen, verifizieren muss.

5 Inverse-rule Algorithmus

Die Idee des inverse-rule Algorithmus [DG97] ist, dass man zunächst zu den einzelnen Views Regeln berechnet, die diese Views "invertieren". D.h. man berechnet Regeln, wie man Tupel der einzelnen Datenbank-Relationen aus den Views berechnen kann. Diese Regeln werden in einem Preprocessing-Schritt bei der Viewdefinition berechnet.
Für das in Abschnitt 3 dargestellte Beispiel werden folgende Regeln gebildet:

- Flugzeug(x,y):=$V_3(x, y)$
- Flugzeug(x,x):=$V_4(x)$
- Bahn(x,y):=$V_1(x, y)$
- Schiff(x,y):=$V_3(x, y)$
- amMeer(x):=$V_2(x)$
- amMeer(x):=$V_4(x)$
- gleichesLand(x,y):=$V_1(x, y)$

Aus diesen Regeln kann man dann die Views berechnen, die man zur Beschreibung der Anfrage Q benötigt. Allerdings werden Views, die mehrere Zwischenziele enthalten dabei mehrfach verwendet, was zu unnötig "aufgeblasenen" Queries führt, die nicht effizient auszuführen sind.

6 MiniCon Algorithmus

Der MiniCon Algorithmus [PH01] versucht die Schwächen der beiden vorhergehend beschriebenen Algorithmen zu umgehen. Er ist ähnlich dem Bucket Algorithmus in zwei Phasen eingeteilt. Idee dabei ist bereits in Phase 1 die einzelnen Möglichkeiten, die Variablen des Queries auf die des Views abzubilden zu evaluieren, um viele mögliche Lösungen für Phase 2 bereits verwerfen zu können.

Dazu werden sogenannte MiniConDescription (MCD) definiert. Ein MCD ist eine Abbildung eines Teils der Variablenmenge des Querys Q auf die Variablen eines Views V. Zu jedem MCD wird dann die Menge G_c gebildet, die die Teilziele in Q enthält, die durch die im MCD gegebene Abbildung der Variablen abgedeckt sind. Dies bedeutet, dass der View eine Umformulierung des Teilqueries G_c darstellt.

Für das in Abschnitt 3 dargestellte Beispiel werden daher die MCDs wie in

V	ϕ	G_C	
V_1	$a \to x,\ b \to y$	Bahn(a,b)	
V_2	$b \to x$	amMeer(b)	
V_3	-	$\emptyset$	Deckt kein Teilziel ab
V_4	$b \to x$	amMeer(b)	

Tabelle 1. MCDs für das Beispiel Query: Q(a):= Bahn(a,b),amMeer(b)

Tabelle 1 gebildet.
Dabei ist V der betrachtete View, ϕ die betrachtete Abbildung der Variablen des Queries auf die des Views und G_C die dardurch abgedeckten Teilziele.

In der zweiten Phase müssen die einzelnen MCDs dann kombiniert werden, um alle Teilziele zu erfüllen. Aufgrund der Vorarbeit im ersten Schritt reicht es dabei Kombinationen zu prüfen, bei denen die abgedeckten Teilziele G_C paarweise verschieden sind. Dies erspart das Durchprobieren vieler nicht gültiger Lösungen und führt so zu einer schnelleren Bearbeitung. Außerdem verhindert es, dass redundante Umformulierungen der Anfrage entstehen. Im Beispiel führt das zu den Kombinationen, die in Abschnitt 3 diskutiert wurden.

7 Vergleich der Algorithmen

In der komplexitätstheoretischen Analyse lässt sich nicht feststellen, dass der MiniCon Algortihmus eine bessere Laufzeit, als die beiden anderen vorgestellten Algorithmen hat.
Im Worst Case liegt die Laufzeit aller drei hier vorgestellten Algorithmen in $O(n \cdot m \cdot M)^n$ mit n der Anzahl der Teilziele der Anfrage, m der maximalen Anzahl der Teilziele in einer Viewdefinition und M als die Anzahl der Views. Es ist auch nicht zu erwarten, einen wesentlich besseren Algorithmus zu finden, da das dargestellte Problem eine Anfrage mithilfe von Views zu beantworten NP-Schwer ist (vgl. [LTW09]).

Allerdings zeigt sich, dass der MiniCon Algorithmus in der Praxis im vergleich zu den anderen vorgestellten Algorithmen sehr performant ist. Dies wurde in der Veröffentlichung von [PH01] anhand einer Studie der Performance dieser drei Algorithmen illustriert.

Um die Studie durchzuführen, wurden die vorgestellten Algorithmen implementiert und ihre Performance gemessen.
Die Eingaben kamen dabei von einem zufallsbasierten Query-Generator, sodass möglichst viele verschiedenartige Queries abgedeckt wurden. Für jede Anzahl an Views wurde deshalb die Performance bei mehrere hundert Queries gemessen, um einen aussagekräftigen Mittelwert bilden zu können.

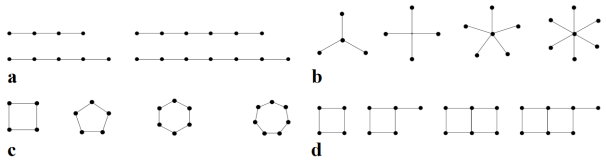

Abb. 1. Verschiedene Arten von Queries. **a** chain; **b** star; **c** cycle; **d** grid [SMK97]

Die Performance wird dabei bei Queries verschiedener Klassen durchgeführt. Die Abb. 1 stellt verschiedene Arten von Queries dar. Die einzelnen Punkte im Graphen stehen dabei für ein Teilziel, die Kanten für (mindestens) eine gemeinsame Variable zwischen den Teilzielen. So ist beispielsweise das Query Q:=Flugzeug(a,b),Bahn(b,c),Schiff(b,d),amMeer(b) ein Star-Query, welches der Form des ersten unter **b** gezeigten Graphen entspricht, wobei Flugzeug(a,b) dem mittleren Knoten entspricht.

7.1 Chain Queries

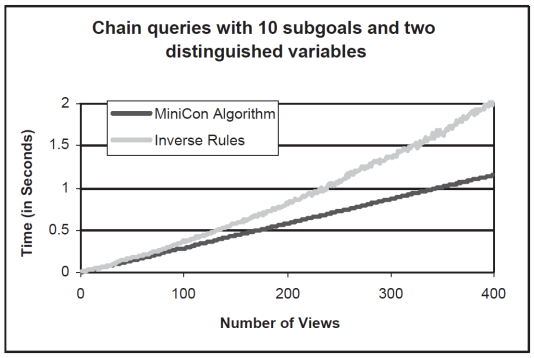

Abb. 2. Vergleich der Performance bei Chain Queries mit 10 Teilzielen und 2 unterschiedlichen Variablen. [PH01]

In Abb. 2 wird die Performance des MiniCon und des inverse-rule Algorithmus bei Chain Queries mit je 10 Teilzielen und zwei verschiedenen Variablen miteinander verglichen. Man erkennt, dass der MiniCon Algorithmus bessere Leistung als der inverse-rule Algorithmus liefert. Der Bucket Algorithmus wird hier nicht mit verglichen, da er eine wesentlich schlechtere Leistung als die beiden anderen Algorithmen liefert. Mit steigender Anzahl Views steigt der Vorteil des MiniCon Algorithmus. Für einen ähnlichen Fall mit unterschiedlichen Viewgrößen zeigt Abb. 3 ein ähnliches Verhalten, wobei der Vorsprung des MiniCon Algorithmus mit steigender Viewzahl hier stärker zunimmt. Die Differenz in der

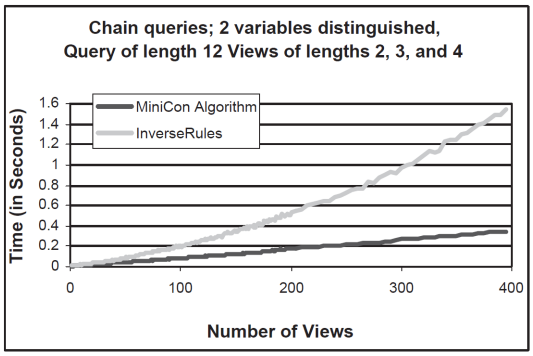

Abb. 3. Vergleich der Performance bei Chain Queries mit 12 Teilzielen und 2 unterschiedlichen Variablen. [PH01]

Performance liegt im Unterschied in der zweiten Phase der Algorithmen. Der MiniCon Algorithmus muss in der zweiten Phase viel weniger mögliche Kombinationen von Views betrachten.

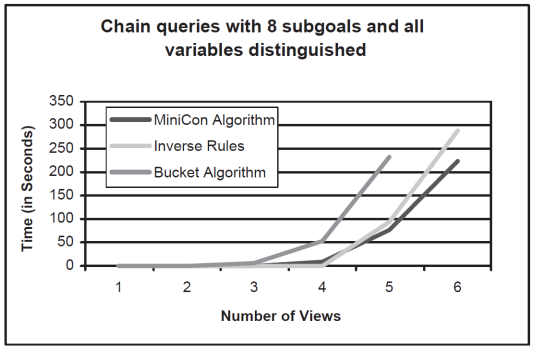

Abb. 4. Vergleich der Performance bei Chain Queries mit 8 Teilzielen und keinen gleichen Variablen. [PH01]

Die Abb. 4 zeigt die Performance, wenn es in der Anfrage keine gleichen Variablen gibt. Die Abbildung zeigt leider nicht, dass die Varianz insbesondere beim inverse-rule Algorithmus in diesem Fall sehr hoch ist.

7.2 Star and complete queries

In Abb. 5 wird die Performance der Algorithmen bei Star-Queries verglichen. Man erkennt, dass der MiniCon Algorithmus ab einer gewissen Anzahl Views wesentlich besser als die anderen Algorithmen performt. Diese bessere Performance erklärt sich erneut in einer wesentlich schnelleren zweiten Phase, da der

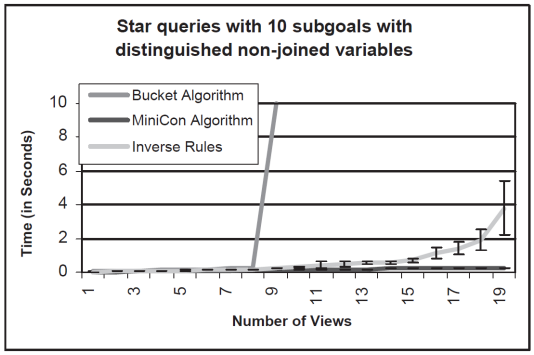

Abb. 5. Vergleich der Performance bei Star-Queries mit 10 Teilzielen [PH01]

MiniCon Algorithmus dort wesentlich weniger Kombinationen der Views durchprobieren muss.
Auch im Falle von vollständigen Queries ist der MiniCon Algorithmus, wie in Abb. 6 gezeigt, den beiden anderen Algorithmen überlegen. Vollständige Queries sind Queries, bei den jedes Teilziel in Abb. 1 mit jedem anderen verbunden ist, also mindestens eine gemeinsame Variable hat.

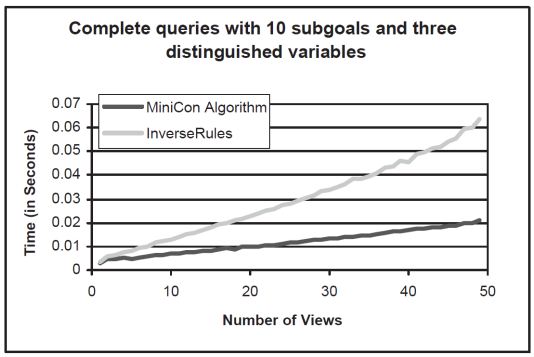

Abb. 6. Vollständige Queries mit 10 Teilzielen und 3 unterschiedlichen Variablen [PH01]

8 Ausblick und Fazit

Die Arbeit hat drei verschiedene Algorithmen zur Querybearbeitung mit Views vorgestellt: den Bucket Algorithmus, den inverse-rule Algorithmus sowie den MiniCon Algorithmus. Obwohl alle drei Algorithmen in der (kompläxitäts-)theoretischen Betrachtung die gleiche Laufzeit haben, so stellt sich heraus, dass der MiniCon Algorithmus in der Praxis deutlich bessere Performance liefert. Das hängt im Wesentlichen damit zusammen, dass in der zweiten Phase weniger mögliche Kombinationen von Views betrachtet werden müssen.

[PH01] erwähnt auch, dass der MiniCon Algorithmus auch für das Problem, wie die Anfragen am besten optimiert werden können um die Systemlast zu reduzieren, sehr einfach angepasst werden kann. Auch in dieser hier nicht weiter behandelten Problemstellung der kostenbasierten Umformulierung von Anfragen (z.B. nach [CKPS95]) liefert der MiniCon Algorithmus eine bessere Performance, als die anderen beiden vorgestellten Algorithmen.

Literatur

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 190–200. IEEE, 1995.

[CM77] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[DG97] Oliver M Duschka and Michael R Genesereth. Answering recursive queries using views. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 109–116. ACM, 1997.

[Hal01] Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[LTW09] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic el using a relational database system. In *IJCAI*, volume 9, pages 2070–2075, 2009.

[PH01] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *The VLDB Journal—The International Journal on Very Large Data Bases*, 10(2-3):182–198, 2001.

[RLJ96] Anand Rajaraman, Alon Y Levy, and J Ordill Joann. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases, VLDB-96, Bombay, India*, 1996.

[SMK97] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal—The International Journal on Very Large Data Bases*, 6(3):191–208, 1997.