Heuristiken zur regelbasierten Optimierung

Joachim Kröger, Stefan Paul, Andreas Heuer

Universität Rostock
Fachbereich Informatik
Lehrstuhl Datenbank– und Informationssysteme
18051 Rostock

{jo, sp, ah}@informatik.uni-rostock.de
http://wwwdb.informatik.uni-rostock.de/Forschung/CROQUE.html

1 Motivation: Regelbasierte Optimierung

Das CROQUE–Projekt[1] ([HK96, SGG+97]) beschäftigt sich mit verschiedenen Aspekten der Optimierung objektorientierter Anfragen. Ausgangspunkt sind Anfragen in ODMGs OQL ([Cat96], formalisiert in [RS97]), die zunächst intern in einem hybriden Ansatz aus Kalkül und Algebra dargestellt werden ([GKG+97]) und dann mittels eines regel– und kostenbasierten Optimierers umgeformt werden sollen. Die kostenbasierte Komponente besteht dabei aus einem bereits entwickelten Kostenmodell ([Jan97]). Die Konzeption der regelbasierten Komponente (und damit auch Teile der Steuerung der Optimierung) wird hier beschrieben. Diese Beschreibung ist vollkommen unabhängig vom CROQUE–Projekt selbst; alle aufgeführten Heuristiken lassen sich somit prinzipiell für beliebige regelbasierte Optimierer nutzen.

Regelbasierte Optimierer und regelbasierte Optimierer–Generatoren haben zunächst trivialerweise *eine* Gemeinsamkeit: Sie basieren auf mindestens einer Regelmenge, deren Regeln erlaubte Umformungsschritte (Äquivalenzen) formulieren. In Abbildung 1a steht jeder Kreis für eine Regel. Bei dieser einen Gemeinsamkeit bleibt es aber meistens auch: Die Verwendung der Regeln geschieht in unterschiedlicher Art und Weise, z.B. können Anfragen top down oder bottom up ausgewertet werden, die Regelmenge kann in kleinere Regelmengen aufgesplittet sein, die in gewissen Reihenfolgen verwendet werden, alle Regeln können erschöpfend ("exhaustive") angewendet werden oder die Verwendung von Regeln kann auf verschiedene Arten heuristisch eingeschränkt sein, usw.

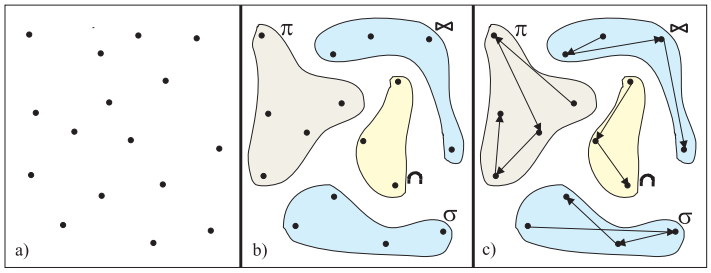

Abbildung 1: Unterschiedliche Regelorganisationsformen

Das Merkmal *regelbasiert* liefert als Hauptvorteil im allgemeinen *Erweiterbarkeit*, beinhaltet aber nicht automatisch auch gleichzeitig *Effizienz*. Betrachtet man neuere (ohne Beschränkung der Allgemeinheit: algebraische) Sprach– und System–Entwicklungen, so kann man im allgemeinen feststellen, daß ein Trend hin zu semantisch einfacheren Operatoren geht, da deren Optimierung (bzw. die Formulierung von Äquivalenzen für diese Operatoren) *erfolgversprechender* und *einfacher* (mithin auch effizienter) erscheint, als die Optimierung komplexer Operatoren. Auf diese Weise entstehen Algebren wie KOLA ([CZ96, Che96]), die aus ca. 60 Operatoren mit 660 Äquivalenzregeln besteht. Der naive Ansatz, alle gegebenen Äquivalenzregeln gegen jeden Operatorknoten im zu optimierenden Anfragebaum auf Anwendbarkeit hin zu testen ("matchen"), führt sichtlich schon bei relativ kleinen Anfragen zu immens großen Anzahlen an Matching–Versuchen. Dessen ungeachtet werden auch die Suchräume äquivalenter, durch erfolgreich gematchte Regeln erzeugter Anfragen schnell sehr groß. Diese Beobachtungen motivieren die Verwendung von Heuristiken zur regelbasierten Optimierung im hier beschriebenen Ansatz mit dem Ziel einer effizienten, regelbasierten Optimierung.

[1] CROQUE (*C*ost– and *R*ulebased *O*ptimization of object–oriented *QUE*ries) ist ein Kooperationsprojekt der Universitäten Rostock und Konstanz. CROQUE wird von der DFG gefördert unter den Aktenzeichen He 1768/5–2 und Scho 554/1–2.

Im folgenden Abschnitt 2 werden die Ansätze verschiedener Systeme zur regelbasierten Optimierung kurz betrachtet. Anschließend wird in Abschnitt 3 die Verwendung von Heuristiken im Rahmen der regelbasierten Optimierung diskutiert, bevor in Abschnitt 4 auf die Regelsortierung in CROQUE — als die prominenteste Heuristik dieses Projektes — ausführlicher eingegangen wird. Wir schließen mit einer kurzen Zusammenfassung und einem Ausblick.

2 Ansätze verschiedener Systeme

COKO–KOLA ([CZ98]) ist die Kombination der bereits erwähnten Algebra KOLA ([CZ96, Che96]) mit der Sprache COKO zur Steuerung von Transformationen. COKO stellt einen Rahmen zur Programmierung von Regelanwendungen und Anfragebaum–Durchläufen bereit. Nachteilig ist die Programmierung der Regelreihenfolgen und die dadurch fehlende Flexibilität.

Im Optimierer–Generator Volcano wird die logische Dimension des Suchraums zunächst vollständig ([McK93]) oder "on demand" ([GM93]) aufgespannt. Die Erzeugung der physischen Dimension kann (zumindest gemäß [GM93]) vom Datenbank–Implementierer mittels einer *promise*–Funktion gesteuert werden. Eine zielgerichtete Suche, die gewünschte physische Eigenschaften zusichert, verhindert dabei die Anwendung nicht zur Lösung der gestellten Anfrage beitragender Regeln. Verschiedene Tuningmechanismen wie branch–and–bound minimieren den Aufwand des Verfahrens erheblich. Nachteile sind aus unserer Sicht die Programmierung der promise–Funktion (sowohl die Tatsache, als auch die Notwendigkeit dazu) und die Idee der online–Sortierung der jeweils möglichen physischen Umformungsschritte ("moves") durch die promise–Funktion.

In Volcanos Vorgänger EXODUS ([GD87]) wird die Regelauswahl mittels *expected cost factors* berechnet. Gemäß der Kritik in [McK93] werden dabei größere Pläne aufgrund größerer Kosteneinsparungen potentiell bevorzugt; außerdem explodieren die Optimierungszeiten bei steigender Komplexität, so daß der Optimierer gerade für komplexere Anfragen im praktischen Einsatz nicht effizient genug und somit nicht einsetzbar ist. Auch hier ist die online–Regelbewertung aus unserer Sicht einer der Hauptnachteile.

Der relationale Optimierer von Starburst ([HFLP89, PHH92]) erlaubt die Unterteilung der Regelmenge in Klassen, wobei die Klassen vom Datenbank–Implementierer (DBI) nach angestrebten Zielen gebildet werden (z.B. predicate migration, selection push–down). Die Regeln der Klassen werden sequentiell oder nach einem *priority scheme* angewandt. Sowohl die Programmierung der Regelklassen und der Regelreihenfolge durch den DBI als auch die Klassifikation der Regeln nach Strategien nehmen dem System entscheidende Flexibilität.

Grundlage des Regelinterpreters von GOM ([KM90]) ist, wie auch bei Starburst, die Bildung von Regelgruppen, die Regeln gleicher Intention zusammenfassen. Die Regeln können innerhalb dieser Gruppen auch nach ihrer Güte geordnet werden. Alle Regelgruppen bilden ein Netz, durch das die Aufeinanderfolge der Verwendung der Gruppen sehr flexibel gesteuert werden kann, um die Verwendung der physischen Pfadindizes ("Access Support Relations") zu unterstützen. Nachteilig ist aber auch hier die feste Programmierung der Regel– und Gruppen–Reihenfolgen.

Gral ([BG92]) erlaubt Regelmengen, die gemäß einer von drei vorgegebenen Strategien auf Anfragen angewendet werden. Hauptproblem ist dabei die fehlende Trennung von logischem und physischem Rewriting, wodurch immer nur genau ein Plan für weitere Optimierungsschritte vorliegt. Die drei Strategien beinhalten verschiedene Heuristiken zur Auswahl des weiter zu betrachtenden Plans: die Anwendung von Regeln gemäß der Ordnung im Regelfile, die Auswahl des Ausdrucks, zu dessen Erzeugung die meisten Regeln angewandt wurden sowie eine Kostenbewertung eines (physischen) Plans. Diese Heuristiken finden sich auch in der CROQUE–Konzeption wieder; ihre Verwendung in Gral ist aber zu sehr eingeschränkt.

3 Verwendung von Heuristiken

Wir gehen der Vollständigkeit halber zunächst vom naiven Ansatz aus Abschnitt 1 aus: *Alle* gegebenen Äquivalenzregeln sollen gegen *jeden* Operatorknoten im zu optimierenden Anfragebaum auf Anwendbarkeit hin getestet werden.

Generell sind Äquivalenzregeln in zwei Richtungen anwendbar. Da aber im allgemeinen eine zumindest heuristisch günstigere, bevorzugte Anwendungsrichtung existiert, betrachten wir im Rahmen des CROQUE–Projektes lediglich *Transformations*regeln, also gerichtete Äquivalenzregeln, die nur in einer Richtung angewendet werden. Wir verwenden im weiteren beide Terme und auch den Begriff "Regel" synonym für "Transformationsregel".

Als Ausgangspunkt der weiteren Betrachtungen beschränken wir uns auf algebraisches Rewriting und die Existenz genau *einer* Regelmenge (gemäß Abbildung 1a). In CROQUE findet der notwendige rekursive Baumdurchlauf außerhalb des allgemein formulierten Pattern–Matchers (der neben algebraischem

Rewriting auch für Kalkül–Rewriting Verwendung findet) statt. Der Pattern–Matcher wird mit Paaren von (Teil–) Anfragebäumen und Regeln aufgerufen, wobei die Regeln jeweils gegen den Wurzelknoten des betrachteten (Teil–)Baums gematcht werden sollen. Die skizzierte "vollständige Suche" macht hier kaum Sinn, da die meisten Regeln von vornherein gegen die meisten (Teilbaum–Wurzel–)Knoten nicht erfolgreich matchen können. Hier läßt sich auf verschiedene Arten die Effizienz erhöhen:

- Der scheinbar günstigste Fall: Alle Regeln werden in einer Hash–Tabelle abgelegt. Der betrachtete (Teil–)Baum wird gehasht (gemäß des Auftretens von Operatorknoten, der Baumstruktur) und nur Regeln, die über die Hash–Tabelle gefunden werden, können überhaupt matchen, sind also anwendbar. Der Nachteil dieses Verfahrens wird jedoch sehr bald augenscheinlich: Die Hash–Funktion ist einerseits im allgemeinen Fall nicht trivial und verlangt andererseits vor allem auch die Betrachtung *tieferliegender* Knoten des Anfragebaums um geeignete Regeln aufzufinden. Somit ist dann die Hash–Funktion gleichzeitig auch ein Pattern–Matcher. Eine offene Fragestellung bleibt die Wahl der Hash–Funktion. Ist sie überhaupt geschlossen darstellbar? Wieviele Knoten des Anfragebaums müssen betrachtet werden? Wir behandeln diesen Ansatz hier nicht weiter.

- In CROQUE ist zwar die Verfügbarkeit allgemeiner Hash–Funktionalität geplant, ein Pattern–Matcher steht jedoch bereits jetzt zur Verfügung. Daher betrachten wir zunächst eine weitere Möglichkeit: Die Regelmenge wird in Regel–Klassen aufgeteilt, wobei der Wurzelknoten der linken Regelseite (des zu matchenden Ausdrucks) die Klassen identifiziert. Die Klassifizierung nach Wurzel–Operatoren ist in Abbildung 1b angedeutet. Ein Hashing aller Regeln ist somit möglich, die benötigte Hash–Funktion sehr einfach, und durch einmaligen Aufruf dieser Funktion kann die Menge der zu betrachtenden Regeln für jeden (Teil–)Anfragebaum bereits auf eine Regelklasse eingeschränkt werden. Die Einfachheit des Verfahrens und die für eine Implementierung erreichbare Effizienz geben den Ausschlag für die Wahl dieses Verfahrens.

Der Leitgedanke, die einzelnen Module und Komponenten möglichst einfach zu halten, stammt aus der Entwicklung des Kostenmodells ([Asm95, Jan97]), wo wir auch auf gewisse Genauigkeiten verzichtet haben, um dafür eine insgesamt höhere Effizienz der Kostenberechnung an sich zu erreichen (schließlich ging es dort nicht um eine möglichst exakte Kostenschätzung, sondern lediglich darum, Vergleichbarkeit von Plänen herzustellen).

Die gebildeten Regelklassen sind noch nicht die endgültige Lösung: Eine Klasse bildet eine *Menge* von Regeln, wobei der Pattern–Matcher aber immer nur eine Regel zur Zeit anwenden kann (Parallelisierung ist an dieser Stelle zunächst nicht geplant). Wir streben daher im nächsten Abschnitt eine Sortierung der Regeln innerhalb der jeweiligen Klassen an und erhalten dann für jede Klasse eine *Liste* von Regeln.

4 Regelsortierung in CROQUE

Anfragesprachen mit vielen Operatoren und sehr vielen Transformationsregeln erzeugen für komplexe Anfragen potentiell sehr viele äquivalente Anfragen. Wenn neben der logischen auch eine physische Dimension aus äquivalenten Algorithmen (wie etwa in CROQUE in Abhängigkeit von physischen Organisationen, z.B. Indizes, Replikation, materialisierten Views [GGMS97], usw.) betrachtet wird, können dabei Suchräume entstehen, die nur mittels Suchverfahren durchsucht werden können oder schon bei ihrer Erzeugung durch die Verwendung von Heuristiken eingeschränkt werden müssen, um eine effiziente Optimierung erreichen zu können.

Viele Suchverfahren arbeiten mit Zufallsstartwerten und besuchen dann Pläne auf Grundlage der ermittelten Kosten des aktuellen Plans, des bisherigen Kostenminimums und der vorangegangenen Schritte. Im Rahmen von CROQUE wurde bei der Untersuchung von Suchverfahren ([Ros96, Ill96]) versucht, die zufällige Wahl eines Ausgangsplans und die vorliegende Sortierung der Pläne dahingehend zu beeinflussen, daß möglichst schnell möglichst gute Pläne gefunden werden. Zu einer Sortierung des Suchraums, die aufgrund der Erzeugung der Pläne automatisch geschah, wurde die sehr einfache Heuristik verwendet, daß Anfragepläne, zu deren Erzeugung mehr logische Transformationsregeln benötigt wurden, besser sind als Pläne, die in weniger Schritten erzeugt wurden. Diese Heuristik ist begründet in der Entscheidung, nur gerichtete Regeln (Transformationsregeln) zu verwenden, die nur in der heuristisch günstigeren Richtung angewendet werden. Je häufiger demnach eine Regelanwendung (Verbesserung) eintritt, desto besser ist auch der Ergebnisplan.

Diese Heuristik betrachtete jedoch nur die Quantität und nicht die Qualität (die "Güte") der angewendeten Regeln und lieferte daher nicht immer befriedigende Ergebnisse, obwohl durchaus Trends in Richtung auf eine Bestätigung der Heuristik erkennbar waren.

Aus dieser Beobachtung und der Notwendigkeit einer (heuristischen) Suchraumeinschränkung leitet sich eine andere Heuristik ab, die im Mittelpunkt der derzeitigen Untersuchungen in CROQUE steht:

Es sollen pro Rewriting eines (Teil–)Anfragebaumes nur die "n besten" Regeln angewandt werden. Dazu werden die Regeln sortiert. Die Sortierung ist in Abbildung 1c skizziert. Mit der Sortierung wird bei dieser Strategie gleichzeitig eine Minimierung der Anzahl von Aufrufen des Pattern–Matchers erreicht. Auch andere Ansätze sehen eine Ordnung der Regeln vor, wobei diese aber im allgemeinen programmiert werden muß (s. Abschnitt 2).

Um den Overhead möglichst gering zu halten, verzichten wir auf eine online–Sortierung während der Anfragebearbeitung. Die Regeln werden also während des Rewritings einfach in der Reihenfolge ihres Auftretens in der Liste einer Regel–Klasse verwendet. Die Regeln werden einmalig initial sortiert, und anschließend wird die Sortierung in Adaptions–Schritten modifiziert. Diese Schritte können wahlweise nach gewissen Zeitintervallen, zu gewissen Zeitpunkten, nach vielen oder bedeutenden Datenbank–Updates, nach vielen Anfragen oder jedem beliebigen anderen Verfahren offline durchgeführt werden. Durch die Adaption sollen evtl. aufgetretene Fehler oder Ungenauigkeiten der initialen Sortierung behoben und die Sortierung zusätzlich (längerfristig) an die beobachtete Anfragelast angepaßt werden.

Die initiale Sortierung der Regeln wird nach einem sehr simplen Verfahren durchgeführt: Zwei Regeln sind vergleichbar (können in eine eindeutige Reihenfolge gebracht werden), wenn es ein kleinstes gemeinsames Muster gibt, auf das beide Regeln anwendbar sind. Aus der Anwendung beider Regeln auf das gleiche Muster läßt sich meist sehr leicht "per Hand" ein Ranking vornehmen. Die Sortierung wird darüber hinaus als transitiv angenommen. Unvergleichbare Regeln werden bei der initialen Sortierung zufällig angeordnet. Zur Adaption der Sortierung wurde zunächst das folgende Verfahren diskutiert:

- Auswertung aller letztendlich zur tatsächlichen Berechnung ausgewählten Anfragen. Dabei Betrachtung der zur Erzeugung dieser Anfragen verwendeten Regeln. Ranking der Regeln auf Grundlage der beobachteten Häufigkeiten.

Der entscheidende (zumindest theoretische) Nachteil besteht bei diesem Verfahren darin, daß sich eine Fehlentscheidung beim initialen Ranking "aufschaukeln" kann. Das initiale hohe Ranking einer Regel sorgt für häufige Anwendung der Regel und damit für ein erneutes hohes Ranking. Das Verfahren scheint allein aufgrund der Gefahr dieser Art der Fehlerfortpflanzung zunächst als nicht geeignet, obwohl eine Implementation durch Einfügung zusätzlicher Unschärfen das Problem vermutlich "umgehen" könnte. Dieser Ansatz wird im Moment nicht weiter verfolgt. Aufgrund dieses Problems wird zur Zeit für die Adaption der Sortierung das folgende Verfahren favorisiert:

- Verwendung einer Mischung aus Benchmarks und beobachteten Nutzeranfragen zur offline–Berechnung. Dabei können entweder ganze Lösungsräume oder Teilräume berechnet und die zur Erzeugung der tatsächlichen oder gefundenen "besten" Lösung verwendeten Regeln gemäß ihrer Häufigkeit gerankt werden. Wenn lediglich eine Berechnung von Teilräumen durchgeführt werden kann oder soll, ist darauf zu achten, daß beispielsweise mittels unterschiedlicher Läufe verschiedener Such– oder Zufallsverfahren eine möglichst große Vielfalt (nicht nur benachbarter Pläne) untersucht wird. Auf diese Weise kann zumindest die oben beschriebene Fehlerfortpflanzung nahezu vollkommen ausgeschlossen werden.

Die Adaption der Sortierung ist Gegenstand weiterer Untersuchungen im Rahmen des Projektes.

5 Zusammenfassung und Ausblick

Neben einer kurzen Darstellung der regelbasierten Optimierung in anderen Ansätzen wurden folgende Methoden zur Steigerung der Effizienz regelbasierter Optimierung diskutiert: Verwendung von Transformationsregeln (gerichteten Äquivalenzregeln), Hashing der Regeln in Verbindung mit Hashing der betrachteten Anfrage und in Verbindung mit Hashing nur des betrachteten Wurzelknotens (Bildung von Regelklassen) sowie initiale Sortierung und Adaption der Sortierung von Regeln und eine Heuristik zur Ausnutzung eben dieser Sortierung.

Eine vollständige Implementierung dieser Konzepte in CROQUE, die die Evaluierung der Konzepte ermöglichen würde, steht derzeit noch aus, da der Prototyp noch nicht vollständig operational ist. Aufbauend auf der Langfassung ([Pau97]) der hier diskutierten Konzepte werden dann weitere Arbeiten zur Verbesserung der dargestellten Heuristiken folgen. Vor allem die Adaption der Regel–Sortierung ist Gegenstand weiterer Untersuchungen. Ein möglicher Zielpunkt ist eine Regelordnung, in der allen Regeln ihr jeweiliges Optimierungspotential in Form eines (Integer–)Zahlenwertes zugeordnet wird, welcher aufgrund von Anfragelast–Statistiken automatisch adaptiert werden kann.
Eine Langfassung dieses Beitrages ist als [KPH98] verfügbar.

Acknowledgements: Wir danken unseren Konstanzer Projektpartnern Marc H. Scholl, Torsten Grust

und Dieter Gluche für Diskussionen und Hinweise zur Verbesserung der dargestellten Konzepte und unseren Studenten Ralf Asmus, Regina Illner, Holger Janz, Stefan Paul, Beate Porst, Denny Priebe und Steffen Rost für die Ausgestaltung und Implementierung der verschiedenen Konzepte in den ersten Prototypen.

Literatur

[Asm95] Asmus, R.: Konzept und Implementierung einer optimierten ODMG–OQL–Schnittstelle auf Object-Store. Diplomarbeit, Universität Rostock, Fachbereich Informatik, Juni 1995.

[BG92] Becker, L.; Güting, R.: Rule-Based Optimization and Query Processing in an Extensible Geometric Database System. In: *ACM Transactions on Database Systems, Vol. 17, No. 2*, S. 247–303, Juni 1992.

[Cat96] Cattell, R. (Hrsg.): *The Object Database Standard: ODMG-93, Release 1.2*. Morgan-Kaufmann, San Mateo, CA, 1996.

[Che96] Cherniack, M.: Form(ers) Over Function(s): The KOLA Reference Manual. Technischer Bericht, Department of Computer Science, Brown University, Providence, RI, Juni 1996. TR in Progress.

[CZ96] Cherniack, M.; Zdonik, S.: Rule Languages and Internal Algebras for Rule-Based Optimizers. In: *Proc. of the ACM SIGMOD Int. Conference on Management of Data, Montreal, Quebec*, Juni 1996.

[CZ98] Cherniack, M.; Zdonik, S.: Changing the Rules: Transformations for Rule–Based Optimizers. In: *Proc. of the ACM SIGMOD Int. Conference on Management of Data, Seattle, WA*, Juni 1998. Accepted Paper. To Appear.

[GD87] Graefe, G.; DeWitt, D.: The EXODUS Optimizer Generator. In: *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, S. 160–172. San Francisco, CA, USA, Mai 1987.

[GGMS97] Gluche, D.; Grust, T.; Mainberger, C.; Scholl, M.: Incremental Updates for Materialized Views with User-Defined Functions. In: *Proc. of the Fifth Int. Conference on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland*, Dezember 1997.

[GKG+97] Grust, T.; Kröger, J.; Gluche, D.; Heuer, A.; Scholl, M.: Query Evaluation in CROQUE - Calculus and Algebra Coincide -. In: *Proc. of the 15th British National Conference on Databases (BNCOD 15), London, UK, LNCS 1271, Springer*, S. 84–100, Juli 1997.

[GM93] Graefe, G.; McKenna, W.: The Volcano Optimizer Generator: Extensibility and Efficient Search. In: *IEEE Conference on Data Engineering 4/1993*, S. 209–218. Vienna, Austria, April 1993.

[HFLP89] Haas, L.; Freytag, J.; Lohman, G.; Pirahesh, H.: Extensible Query Processing in Starburst. In: *SIGMOD Record, 18 (1)*, S. 377–388, März 1989.

[HK96] Heuer, A.; Kröger, J.: Query Optimization in the CROQUE Project. In: *Proc. of the 7th Int. Conf. on Database and Expert Systems Applications (DEXA '96), Zurich, Switzerland, LNCS 1134, Springer*, S. 489–499, September 1996.

[Ill96] Illner, R.: Verwendung von Simulated Annealing zur kostenbasierten Optimierung objektorientierter Anfragen. Diplomarbeit, Universität Rostock, Fachbereich Informatik, Juni 1996.

[Jan97] Janz, H.: Anpassung des CROQUE-Kostenmodells an die flexible interne Ebene. Diplomarbeit, Universität Rostock, Fachbereich Informatik, Juli 1997.

[KM90] Kemper, A.; Moerkotte, G.: Advanced Query Processing in Object Bases Using Access Support Relations. In: *Proc. of the 16th VLDB Conference*, S. 290–301. Brisbane, Australia, 1990.

[KPH98] Kröger, J.; Paul, S.; Heuer, A.: Über die Verwendung von Heuristiken zur regelbasierten Optimierung von Datenbank-Anfragen. Rostocker Informatik-Bericht 21/1998, Universität Rostock, Fachbereich Informatik, 1998.

[McK93] McKenna, W.: *Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer*. Dissertation, University of Colorado, 1993.

[Pau97] Paul, S.: Analyse des Optimierungspotentials der algebraischen Äquivalenzregeln in CROQUE. Diplomarbeit, Universität Rostock, Fachbereich Informatik, Dezember 1997.

[PHH92] Pirahesh, H.; Hellerstein, J.; Hasan, W.: Extensible/Rule Based Query Rewrite Optimization in Starburst. In: *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, S. 39–48, Juni 1992.

[Ros96] Rost, S.: Analyse alternativer Suchstrategien zur kostbasierten Optimierung objektorientierter Anfragen. Diplomarbeit, Universität Rostock, Fachbereich Informatik, Juni 1996.

[RS97] Riedel, H.; Scholl, M.: A Formalization of ODMG Queries. In: *Proc. of the 7th Int. Conference on Database Semantics (DS-7), Leysin, Switzerland*, Oktober 1997.

[SGG+97] Scholl, M.; Gluche, D.; Grust, T.; Heuer, A.; Kröger, J.: Optimierung von generischen Operationen und Speicherstrukturen in Objekt-Datenbanken (Arbeitsbericht). Preprint CS–05–97, Fachbereich Informatik, Universität Rostock, April 1997. **Gleichzeitig erschienen als:** Technical Report 32/1997 (Konstanzer Schriften in Mathematik und Informatik, Nr. 32), Fakultät für Mathematik und Informatik, Universität Konstanz.