

A Tractable Query Cache By Approximation

Daniel Miranker¹, Malcolm C. Taylor², and Anand Padmanaban³

¹ Department of Computer Sciences, University of Texas
Austin, Texas 78712

² Radiant Networks, Cambridge, UK

³ Oracle Corp., Redwood City, CA

`miranker@cs.utexas.edu`

`malcolm.taylor@radiantnetworks.co.uk`

`Anand.Padmanaban@oracle.com`

Abstract. In this paper we present the organization of a predicate-based query cache suitable for integration with agent-based heterogeneous database systems. The cache is managed using a tractable (*sound and complete*) query containment algorithm, yet there are no language restrictions placed on the applications. This is accomplished by introducing query approximation.

Query approximation is a compilation technique where a query expression in a general query language is mapped to a query expression in a restricted language. We define a target language such that query containment can be tested in polynomial time. We specify the algorithms by which the query engine and the cache manager may negotiate a choice of approximation and the development of a query plan. We use two application workloads and the TPC-D benchmark queries to assess the value of the cache within the architecture of the InfoSleuth heterogeneous database system.

1 Introduction

The speed by which heterogeneous databases may gather data from component databases suffers from additive network and database latencies. One active approach to ameliorating this cost is to cache previous queries and/or to maintain materialized views (prefetching) of the component databases. A central element in these approaches is to compute query containment. That is given a pair of queries, Q_1 and Q_2 , the first derived from an application, the second a representation of a data source, is the data comprising the answer set of Q_1 contained in (subsumed by) the data in the answer set to Q_2 . If so then the answer set of Q_2 may be used to compute the answer set of Q_1 . We would like to determine if Q_2 contains Q_1 , written as $Q_2 \supseteq Q_1$, by examining only the text of the query predicates and not rely on the materialization of the answer set or other physical properties (e.g. an address of a disk page).

The technical challenge for predicate caching as well as dealing with component DBs as views is that if the queries are expressed in any common query

language, determining query containment is NP-hard. There have been many efforts to develop query languages where query containment is tractable [8, 12, 10, 3, 16]. Although a number of useful results have been obtained, these languages place severe restrictions on the select and/or join conditions allowed in queries, and hence have limited applicability to the problem of predicate caching.

We define a query language that places few restrictions on the select/join conditions, while still allowing a tractable containment algorithm. We present a cache architecture which does not put any restriction on an application's queries, yet maintains a query cache based on tractable query containment. The basic idea is that there are precise syntactic structures that make query containment hard. In our system an application query is checked to see if it contains such a construct. If so, a new query is formulated by removing the problematic construct. The new query is a *weakened approximation* of the original. Thus, the new query is guaranteed to contain the original and evaluating query containment tractable. Only at the final stage of query processing are the omitted predicates applied, reducing the answer set to that of the original query.

This work was done as part of the Infosleuth mediator-based heterogeneous database system [9]. Since, in mapping queries to multiple data sources a mediator must resolve multiple sources of information and optimize cost, the integration of approximation fits naturally. If a query must be approximated there may be a number of ways to generalize the query, each with its own cost and intersection with cached queries. Further, if only some of the data needed to solve a query are cache resident, there can be a cost-based decision to determine whether the cached data should be merged with additional data or if the entire query should be reexecuted. In the spirit of agent architectures query plans are negotiated between the query agent and the cache agent.

Our hypothesis is that in real applications very few queries will require weakening and for those that do, the weakening does not appreciably add to the volume of data processed by the system. We support this hypothesis by examining two application workloads and TPC-D query benchmark and measuring the volume of additional data resulting from the weakening of those queries. Of the entire test suite, only 3 of 17 TPC-D queries are outside our tractable subset.

2 Tractability and Query Approximation

We exploit a narrow syntactic perspective of decidability wrt query languages. We start with queries in a general query language and identify individual predicates whose removal from the query, Q , yields a new query, Q' expressible in the restricted language. Using terminology developed in knowledge-compilation we call the new query an *approximation* of the original query [13]. Since the approximation was constructed by removing predicates, the approximation is guaranteed to contain the original query, $Q' \supseteq Q$. When this property holds the approximation is called an *upper bound*, designated Q_{UB} .

In our architecture the cached predicates are drawn only from a tractable query language. If an application submits a query that is more expressive than

the tractable language, the cache agent must determine an upper-bound approximation. A common problem formulation in knowledge-compilation is, given an expression in an intractable logic, how hard is it to find a tractable least-upper bound. Determining a least-upper bound is often as hard as finding a direct solution. Since we are embedding this concept in a database system we may exploit the same cost-based infrastructure used to improve search in a query optimizer.

The query containment problem has been widely studied in the context of pure conjunctive queries. A pure conjunctive query corresponds to select-project-join queries of relational algebra, without additional built-in predicates. For pure conjunctive queries the containment problem can be reduced to one of finding a *containment mapping* from the variables of Q2 to those of Q1 [2]. A containment mapping is a mapping from the variables of Q2 to those of Q1, such that each literal in the body of Q2 is mapped to a literal in the body of Q1. The general containment problem has been shown to be NP-complete [2], but several researchers have identified restricted classes of pure conjunctive queries for which the problem can be solved in polynomial time. Saraiya [12] showed that query containment can be determined in linear time if no predicate appears more than twice in the sub-goals of Q1. Qian [10] developed a polynomial-time algorithm for the case where Q2 is acyclic. Subsequently, Qian's result was generalized by Chekuri and Rajaraman [3], to the case where the query width of Q2 is bounded by some integer k ; the acyclic queries being those with query width 1.

In practical applications, queries frequently involve comparisons between a variable and a constant, as well as comparisons between two variables. The most common form of comparison is a test for equality, but inequalities ($<$, \leq , $>$, \geq) and disequations (\neq) can also be used. Consequently, there have been investigations into whether the results concerning pure conjunctive queries can be extended to the case of conjunctive queries with built-in predicates. Klug [7] and van der Meyden [14] considered the effect of including inequalities, and showed that the problem becomes Π_2^p -complete. Zhang and Ozsoyoglu [16] define a normal form for conjunctive queries with inequalities, which divides a query into two parts: a pure conjunctive query and a conjunction of equality and inequality comparisons. Tests for containment can then be made by first looking for containment mappings between the pure conjunctive parts, and then checking containment of the comparison parts.

Kolaitis et al [8] considered the effect of allowing disequations as a built-in predicate. They show that the containment problem is in coNP if no database predicate appears in Q1 more than twice. They also show that the problem remains Π_2^p -complete under the restriction that Q2 is acyclic.

Rosenkrantz and Hunt consider only conjunctions of equality and inequality comparisons (without the pure conjunctive component) and show that, as long as disequations are not allowed, *equivalence* of expressions can be deduced in polynomial time [11]. The family of comparisons considered in their work is more general than those considered by Zhang and Ozsoyoglu, in that it allows for a broad range of select and join conditions.

Chen and Roussopoulos [4] also omit the pure conjunctive component of a query. They do allow all equality and inequality comparisons, disjunctions as well as conjunctions. Their containment algorithm is sound but incomplete.

3 Algorithm for Query Containment

A polynomial query containment algorithm is presented for the simple restriction on conjunctive queries where no database predicate may appear more than once. This restriction does not impact most of the select and join conditions encountered in queries. Hence, most queries maybe approximated exactly. Our construction exploits a normal form similar to that used by Zhang and Ozsoyoglu, but using the more general form of comparisons considered by Rosenkrantz and Hunt. We adapt the approach of Rosenkrantz and Hunt to the problem of containment rather than equivalence. Thus, the containment problem can be solved in polynomial time even when allowing for a quite general class of comparisons involving both equalities and inequalities.

Definition 1. (*Tractable subset of SQL*) A SQL query is in the tractable subset if it can be expressed as a conjunctive query with built-in predicates $=, <, \leq, \geq, >$. where each sub-goal of an approximation is expressed in terms of either a database predicate or a built-in predicate with the following restrictions:

- No database predicate may appear in more than one sub-goal.
- Each comparison involving the built-in predicates must take on of the following three forms:
 1. $\langle \text{variable} \rangle \langle \text{comparison-op} \rangle \langle \text{constant} \rangle$
 2. $\langle \text{variable} \rangle \langle \text{comparison-op} \rangle \langle \text{variable} \rangle$
 3. $\langle \text{variable} \rangle \langle \text{comparison-op} \rangle \langle \text{variable} \rangle + \langle \text{constant} \rangle$

Example 1. The following query is in our tractable subset:

$$q(X2, Y2) \leftarrow p(X1, X2, X3, X4) \wedge s(Y1, Y2, Y3) \wedge o(X1, Y1, Z1) = 'Austin' \wedge Z1 \geq 10 \wedge X4 < X3 + 5$$

The division into a pure conjunctive query and a conjunction of comparisons allows each part of the query to be dealt with separately; first seeking a containment mapping between the pure conjunctive components of the queries, and then examining each conjunction of comparison to establish containment. We show that each of these two parts can be done in polynomial time and we further prove that the algorithm is sound and complete.

Definition 2. (*Normal Form*) A query Q is in normal form if it is expressed as $Q_{PCQ} \wedge Q_{COMP}$, where

- Q_{PCQ} is a pure conjunctive query in which no variable appears more than once.
- Q_{COMP} is a conjunction of comparisons that each take one of the forms " $X \leq c$ " and " $Z \leq Y + c$ ", where X and Y are variables, c is a constant, and Z is either a variable or zero.

Algorithm 1 (*Containment of conjunctive queries*)

1. convert each query to normal form
2. check whether there is a symbol mapping ρ from $Q2$ to $Q1$
3. apply the mapping ρ to the comparisons in $Q2$
4. check whether $Q1_{COMP}$ implies $\rho(Q2_{COMP})$

Lemma 1. *Let Q be a query expressed in our tractable subset of SQL. Then Q can be converted to normal form in polynomial time.*

Proof. Suppose a variable X appears twice in Q_{PCQ} . We rename the second occurrence with a variable name that has not been used before (say, $X1$), and introduce an extra comparison " $X = X1$ " in Q_{COMP} . Treating all repeated variables in this way will ensure that no variable appears more than once in Q_{PCQ} . Next, any comparison of form " $X = Y$ " is replaced by a pair of comparisons " $X \leq Y \wedge Y \leq X$ ". Then any comparison using \geq (or $>$) can be rewritten in terms of \leq (or $<$) by simply moving variables and/or constants from one side of the inequality to the other. Then, any comparison involving $<$ is rewritten in terms of \leq by subtracting ϵ from the right-hand side, where ϵ is a value sufficiently small that no value between the new right-hand side and the old right-hand side can be represented on the computer. Finally, if a comparison has a constant on the left-hand side, the value of that constant is subtracted from each side so that zero becomes the left-hand side. \square

Example 2. Let $Q = E(X,Y), D(Y,Z,W), W > 100$. Then Q can be rewritten as $E(X,Y), D(Y1,Z,W), Y \leq Y1, Y1 \leq Y, 0 \leq W - 100 - \epsilon$

Having translated the queries to normal form, we first look for containment mappings between the pure conjunctive parts of the respective queries. The next two lemmas lead us to the result that our four-step algorithm is both sound and complete.

Lemma 2. *Let $Q1$ and $Q2$ be pure conjunctive queries, where $Q1$ has no repeated predicates. Then there cannot be more than one symbol mapping from $Q2$ to $Q1$.*

Proof. Consider any conjunct of $Q2$. There must be at most one predicate of the same name in $Q1$, since $Q1$ has no repeated predicates. So there is at most one way to map this conjunct of $Q2$. Similarly for all other conjuncts of $Q2$. Therefore, there is at most one symbol mapping from $Q2$ to $Q1$. \square

Lemma 3. (*Zhang and Ozsoyoglu*) *Let $Q1$ and $Q2$ be two conjunctive queries in normal form. Let $\rho_1, \rho_2, \dots, \rho_n$ be all the symbol mappings from $Q2$ to $Q1$. Then $Q2 \supseteq Q1$ iff $n \geq 1$ and $Q1_{COMP} \Rightarrow \rho_1(Q2_{COMP}) \vee \dots \vee \rho_n(Q2_{COMP})$*

Theorem 1. *Let $Q1$ and $Q2$ be two conjunctive queries in normal form. Let $M(Q2, Q1)$ denote the set of all symbol mappings from $Q2$ to $Q1$. If $Q1$ has no repeated database predicates, $Q2 \supseteq Q1 \Leftrightarrow \exists \rho \in M(Q2, Q1). Q1_{COMP} \Rightarrow \rho(Q2_{COMP})$.*

Proof. Follows from the Lemma 2 and Lemma 3. □

We now turn our attention to the conjunctions of comparisons. Rosenkrantz and Hunt [11] established that *equivalence* of conjunctions of comparisons can be determined in polynomial time. We adapt their approach to obtain a similar result for *containment*. Each query is represented as a weighted directed graph, in which there is one vertex for each variable and one vertex to represent zero. For each inequality of form " $X \leq c$ " there is an edge from the vertex of X to the vertex of zero, with weight c . For each inequality of form " $Z \leq Y + c$ " there is an edge from the vertex of Z to the vertex of Y , with weight c .

Lemma 4. (*Rosenkrantz and Hunt*) *If there is a path from the vertex of X to the vertex of Y , and the sum of the weights along the path is c , the comparisons in the predicate imply that $X \leq Y + c$*

Lemma 5. (*Rosenkrantz and Hunt*) *Let c be the weight of the shortest path from the vertex of X to the vertex of Y . Then the expression has a satisfying assignment in which $X = Y + c$.*

Lemma 6. (*Rosenkrantz and Hunt*) *An expression is satisfiable iff its graph has no negative weight cycles.*

Theorem 2. *Let $P1$ and $P2$ be conjunctions of comparisons from queries expressed in normal form. Then $P2 \supseteq P1$ iff for all variables X and Y , if the graph of $P2$ has a path from the vertex of X to the vertex of Y with weight c , then the graph of $P1$ has a path from the vertex of X to the vertex of Y with weight $\leq c$.*

Proof. proceeds by reductio ad absurdum.

Suppose the left-hand side is true but the right-hand side is false. Then there exist variables X and Y such that the shortest path from X to Y in $P2$ has weight d (say), while in $P1$ either there is no path from X to Y or else the shortest path has weight c , where $d < c$. If $P1$ has no such path, we can add an edge from X to Y with weight c (where $c > d$ and c is greater than the negative of the shortest path (if any) from Y to X in $P1$). Then the new $P1$ is still satisfiable (since we have not created a negative weight cycle) and is still contained in $P2$. So now, without loss of generality, we assume that $P1$ has a path from X to Y of length c , where $c > d$. Therefore $P1$ has a satisfying assignment in which $X = Y + c$. But $P2$ implies $X \leq Y + d$. And, since $d < c$, $P2$ is not satisfied by any assignment with $X = Y + c$. Therefore $P1$ is not contained in $P2$.

Suppose the left-hand side is false and the right-hand side is true. Then there exists an assignment that satisfies $P1$ but not $P2$. So we add this assignment to $P2$ and we should get a graph that is not satisfiable. (to add an assignment to a graph, we can take each variable assignment $X = x_1$, and add an edge from X to zero with weight x_1 and an edge from zero to X with weight $-x_1$). Since the new graph is not satisfiable, it must have a negative weight cycle. Let that cycle be $v_1, v_2, \dots, v_n, v_1$. Now, by hypothesis, for each pair of adjacent vertices v_i, v_{i+1} in this cycle, there is in $P1$ a path from v_i to v_{i+1} of lesser weight. By combining these lesser-weight paths between each pair of vertices in the cycle,

we obtain a cycle in P1. Moreover, the total weight of this cycle in P1 must be less than or equal to the weight of the cycle in P2. Therefore P1 has a negative weight cycle, so P1 is not satisfiable. \square

Theorem 3. *Let $Q1$ and $Q2$ be two conjunctive queries with built-in equality and inequality predicates such that $Q1$ has no repeated database predicates and all comparisons, in both $Q1$ and $Q2$, have one of the following forms:*

- $\langle \text{variable} \rangle \langle \text{comparison-op} \rangle \langle \text{constant} \rangle$
- $\langle \text{variable} \rangle \langle \text{comparison-op} \rangle \langle \text{variable} \rangle$
- $\langle \text{variable} \rangle \langle \text{comparison-op} \rangle \langle \text{variable} \rangle + \langle \text{constant} \rangle$

Then the containment of $Q1$ in $Q2$ can be checked in polynomial time.

Proof. We use Algorithm 1 to perform the check (by Theorem 1, this algorithm is sound and complete). The algorithm has four steps:

- By Lemma 1, the first step can be performed in polynomial time.
- The second step involves checking, for each predicate symbol in $Q2_{PCQ}$, whether that predicate symbol also appears in $Q1_{PCQ}$. It is clear that this can be done in polynomial time.
- The third step involves applying a symbol mapping to the variables of $Q2$. This again is clearly a polynomial-time operation.
- The fourth step involves computing the weights of the shortest paths between each pair of vertices. This can be done in cubic time[5].

Since each step is polynomial, the complete algorithm is polynomial. \square

4 Agent-Based Negotiation of Query Plans

We address the architectural separation among the logic-related aspect of query approximation and containment in a manner consistent with cost-based decision making. The architecture encapsulates the logic and implementation of caching and the approximation method in the cache agent. The query planning components of the mediator are strictly responsible for choosing the caching plan. In the spirit of agent-based systems, we specify a distributed algorithm where the two agents negotiate a query plan.

In InfoSleuth application queries are executed as follows. Each resource agent (component database) advertises its existence to a mediator, specifying the capabilities of the agent and the ontological fragments for which it can provide information. User queries, expressed in terms of an ontology, are forwarded to the query agent. The query agent contacts the mediator and exploits the ontology and semantic analysis of query language expressions to obtain a list of the available resource that are capable of answering each subquery. The query agent assembles an evaluation plan, mapping subqueries to individual resources and generating a plan to fuse the results into the complete response. Typically resource agents are distributed over a wide-area network, making data transmission the dominant component of response time. If queries can be answered from

cached data, remote access can be avoided and there is a concomitant reduction in response time.

The negotiation exploits the common breakdown of query optimization among logical planning and physical planning. Thus, negotiation is a two step process. (See Figure 4.) The query agent submits a logical plan to the cache agent. The cache may be able to serve the data for the query but may not be able to answer the precise query. The cache agent responds to the query agent whether it can satisfy the query, in whole or in part. In the case that the cache can not provide a precise answer, the cache agent also informs the query agent that it will have to execute some additional predicates when fusing the data. The cache agent may further provide the query agent with cost metrics with which it may determine the quality of an individual plan and search for alternatives.

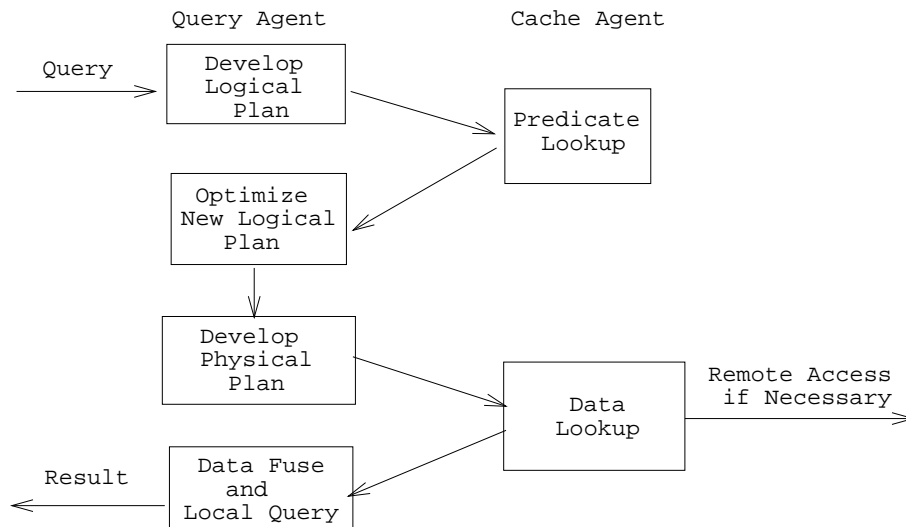


Fig. 1. Negotiation Between Cache Agent and Query Agent

Algorithm 2 (*Negotiation Algorithm*)

The query agent asks, can you service query, Q_i ? Let C represent the content of the cache which is the union of all cached query predicates, $C = \bigcup_j Q_j^{UB}$.

1. Compute Q_i^{UB} . It is expected that most of the time $Q_i = Q_i^{UB}$.
2. Determine if and how Q_i can be satisfied by the cache.
3. Return, 4 cases (see Table 1)

In the simplest case, a request may be satisfiable, completely, from cached data. Thus, given a logical plan, the information returned to the query agent will

	Formal Property	Comment	Reply
a)	$Q_i \cap C = \phi$	No data in the cache that will fulfill the query	Request will be fulfilled by remote request.
b)	$\exists Q_j^{UB} \in C,$ $Q_i = Q_i^{UB} = Q_j^{UB}$	There are data in the cache that precisely fulfills the query.	Request will be fulfilled, precisely, by the cache.
c)	$\exists Q_j^{UB} \in C,$ $Q_j^{UB} \supseteq Q_i^{UB} \supseteq Q_i$	The query is subsumed by a cached query	Request can be fulfilled by the cache, but query agent must execute query predicates, Q' , where $Q' \supseteq Q_i - Q_j^{UB}$
d)	$\exists Q_j^{UB} \in C,$ $Q_i^{UB} \cap Q_j^{UB} \neq \phi$ $\wedge \neg(Q_j^{UB} \supseteq Q_i^{UB})$	Only some of the data needed to fulfill the query are cached.	Request can be partially fulfilled by the cache, by Q_j^{UB} , the remainder fulfilled by querying the resource agent for, $Q'' = Q_i^{UB} - Q_j^{UB}$, the query agent must execute query predicates, Q' , where $Q' \supseteq Q_i - (Q_i \cap (Q_i^{UB} \cup Q_j^{UB}))$

Table 1. Tests for query containment in the cache

be limited to the physical cost of satisfying the logical plan from cache or from a remote access. This comprises cases a) and b). In a more complex situation, the cache agent must also make a determination if only some of the data for the logical request can be answered locally. In that case the logical plan must be decomposed into a pair of new logical plans, one comprising a cache access, the other, a remote access. Since query decomposition is already implemented by the query agent, rather than duplicate this function in the cache agent, the cache agent will suggest to the query agent that it decompose the query accordingly. In essence, unless the query is satisfied by the cache precisely, the query agent and the cache agent negotiate how a query should be executed. The final decision is made by the query agent on a cost-basis. If costs dictate decomposition, the results are fused in precisely the fashion used to fuse any other pair of data sources. These elements comprise cases c) and d).

One obvious case where cost metrics appear likely to by-pass the cache is the execution of a relational select predicate on a table, when its evaluation using the cache comprises a long sequential scan, but its evaluation by the resource agent comprises a clustered index which promptly finds the result.

5 Empirical Results

We identified two query constructs that must be removed from the approximated query. One of these is the disequation predicate (\neq) and the other is the repeated occurrence of a database predicate. Any comparison involving disequation is simply dropped from the query and applied later, after the data have been fetched from the component databases. Our claim is that this will have little affect on performance, because, disequations occur infrequently in practice and

the selectivity of a disequation is typically much greater than that of an equality, or even an inequality, so dropping a disequation from a query is unlikely to cause a substantial increase in data transfer.

In the case of repeated database predicates, our solution is to remove the repeated predicate from the approximated query. A separate query is issued to fetch all the relevant tuples of the repeated predicate and, once the data have been fetched from the resources, a join is performed to compute the result of the original query. The justification for this approach is that

- repeated predicates occur infrequently in practice
- by decomposing the query and caching the separate pieces, we improve the chances of obtaining future cache hits
- in the case of self-join, the operand is usually smaller than the result
- our approach does not preclude the strategy of pushing self-joins down to the resource agents. A cost model in the optimizer may still choose this as the best approach.

In general, a SQL query may contain other constructs that are outside our tractable subset, in particular GROUP BY. The GROUP BY construct itself does not affect the tractability of query containment, but, if the query contains other constructs within the scope of the GROUP BY it will also be necessary to drop the GROUP BY. However, GROUP BY is usually the last operator evaluated in a query, being applied to organize a summary of completed query results. In a heterogeneous database the common use of GROUP BY is to organize the results of the subqueries directed to individual data sources.

The performance of the cache is analyzed with respect to two major applications of InfoSleuth[9]. The EDEN application is a collaboration involving several government organizations in the United States and Europe, enabling the sharing and exchange of environmental information. TechPilot is a competitive intelligence application. TechPilot enables acquiring, integrating and monitoring technical competitive intelligence information from open sources.

To measure the overhead for the approximation scheme we need to consider, per the system’s workload, the frequency in which users’ queries fall outside the tractable subset, and the amount of additional data transferred as a result of weakening the queries. Neither EDEN or the competitive intelligence application contained queries outside the tractable subset. Emboldened by this result we considered the TPC-D benchmark[6]. TPC-D is a standard database benchmark comprising 17 queries intended to reflect the most complicated decision support and data mining applications for data warehouses. Only 3 of the 17 TPC-D queries were outside the subset.

Table 2 illustrates the increase in result size induced by weakening the 3 queries. For query 8F the approximation does not increase the size of the result at all. Query 16F involves both disequation and GROUP BY, which accounts for the increase in result size. Of the three potential workloads only TPC-D query 7F yields a downside to the approach. Query 7F involves a repeated database predicate, each with low selectivity. Generalizing the repeated predicate increases the selectivity and consequently the result size. Further note that in query 7F there

Query Num	Result Size	Size of Approx.
7F	36	916 or 1444
8F	13	13
16F	322	1292

Table 2. Result Size, in rows, for TPC-D Queries and Their Approximations

are two possible weakenings. The negotiation algorithm would report both possible decompositions. A cost-based optimizer could dismiss the cache mechanism entirely.

Per the performance of the cache itself. For EDEN cache hits occur in half the queries, and in the majority of those cases there is a saving of more than 50% in data transfer. For TechPilot, there were cache hits in more than half the queries, and in those cases we found a saving of more than 90% in data transfer.

6 Conclusions and Future Directions

An open question is whether we have defined the most general or even most useful tractable SQL subset. In other words, are there other tractable SQL subsets that may yield better coverage of real-world queries and/or tighter upper-bounds when needed? We reported favorable results, just one questionable query in three different workloads. Nevertheless, a precise way to measure these qualities has not been enunciated.

A next step is the integration of the approach to the mediator itself. A central problem solved by mediators is finding a maximally-contained rewriting of the query in terms of a fixed set of views[6]. In the terminology of knowledge compilation this corresponds to finding the smallest lower-bound cover [13].

Of general interest to the database community is how to enforce correct concurrency control based on logical predicates. It appears that our approach is directly applicable to a lock-based replication scheme proposed by Quass and Widom [15]. But general-purpose extensions are an open issue.

References

1. C. Beeri, A. Levy and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of ACM Symp. on Principles of Database Systems*, 1997.
2. A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. of the ACM Symp. on Theory of Computing*, 1977.
3. C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proc. of Int. Conf. on database theory (ICDT '97)*, pages 16–21, 1997.
4. C.M. Chen and N. Roussopoulos. The implementation and performance evaluation of the ADMS query optimizer: integrating query result caching and matching. Tech. Rep. CS-TR-3159, Computer Science Dept., University of Maryland, 1993.
5. R.W. Floyd. Algorithm 97: Shortest path. *CACM*, 5(6), June 1962.

6. H. Garcia-Molina, J. D. Ullman and J. D. Widom. *Database Systems: The Complete Book, 1/e*. Prentice-Hall, 2002.
7. A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1), January 1988.
8. Ph.G. Kolaitis, D.L. Martin, M.N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *Proc. of ACM Symp. on Principles of Database Systems*, 1998.
9. M. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, A. Unruh. Active Information Gathering in InfoSleuth. *IJCIS* 9(1-2):3-28, 2000,
10. X. Qian. Query folding. In *Proc. of Twelfth Int. Conf. on Data Engineering*, 1996.
11. D.J. Rosenkrantz and H.B. Hunt. Processing conjunctive predicates and queries. In *Proc. of Int. Conf. on Very Large Databases*, 1980.
12. Y. Saraiya. *Subtree elimination algorithms in deductive databases*. Ph.D. thesis, Computer Science Department, Stanford University, 1991.
13. M. Schaerf and M. Cadoli. Tractable reasoning via approximation. In *Artificial Intelligence*, 74, 1995.
14. R. van der Meyden. The complexity of querying infinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1), 1997.
15. D. Quass and J. Widom. On-line warehouse view maintenance. In *Proc. of SIGMOD '97*, 1997.
16. X. Zhang and Z.M. Ozsoyoglu. Some results on the containment and minimization of (in)equality queries. *Information Processing Letters*, (50), 1994.