Conjunctive Query Containment Revisited*

Chandra Chekuri† Anand Rajaraman‡

November 13, 1998

Abstract

We consider the problems of conjunctive query containment and minimization, which are known to be NP-complete, and show that these problems can be solved in polynomial time for the class of *acyclic queries*. We then generalize the notion of acyclicity and define a parameter called *query width* that captures the "degree of cyclicity" of a query: in particular, a query is acyclic if and only if its query width is 1. We give algorithms for containment and minimization that run in time polynomial in n^k, where n is the input size and k is the query width. These algorithms naturally generalize those for acyclic queries, and are of practical significance because many queries have small query width compared to their sizes. We show that good bounds on the query width of Q can be obtained using the *treewidth* of the incidence graph of Q. We then consider the problem of finding an equivalent query to a given conjunctive query Q that has the least number of subgoals. We show that a polynomial time approximation algorithm is unlikely for this problem. Finally, we apply our containment algorithm to the practically important problem of finding equivalent rewritings of a query using a set of materialized views.

1 Introduction

Testing query containment and equivalence are fundamental problems of database theory, and are central to global query optimization in database systems. Conjunctive queries are an important class of database queries, equivalent in expressive power to SPJ queries in the relational algebra. We consider the classical problem of testing containment of conjunctive queries. The problem is

*This work was done while the authors were at Stanford University.

†Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974. Email: chekuri@research.bell-labs.com. Supported at Stanford University by an NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

‡Amazon.com, 1516 2nd Ave, Seattle, WA 98101. E-mail: anand@amazon.com. Supported at Stanford University by an NSF grant IRI–92–23405, ARO grant DAAH04–95–1–0192, and USAF contract F33615–93–1–1339.

well-known to be NP-complete [7]. In view of its practical significance, considerable attention has been devoted to finding classes of conjunctive queries that admit polynomial-time algorithms for equivalence and minimization [1, 2, 12, 4]. *Acyclic queries* in particular have been extensively studied in the context of query optimization in distributed database systems, and are well-known to have desirable algorithmic properties [23].

In this paper, we first present polynomial-time algorithms to test containment of an arbitrary conjunctive query in an acyclic query, and to minimize an acyclic query. We then introduce a new parameter of a query called the *query width,* and show that acyclic queries are precisely the class of queries with query width 1. We generalize the query containment and minimization algorithms to arbitrary queries such that their running time is polynomial in n^k, where n is the input size and k is the query width. These results are significant not only because they naturally generalize the algorithms for acyclic queries, but also because most commonly encountered queries have small query width compared to their sizes. We relate query width to *treewidth,* an extensively studied graph-theoretic parameter, and show that we can obtain good bounds on the query width from the treewidth of the *incidence graph* of the query. We then consider the question of whether there is an efficient (polynomial time) algorithm to approximate the minimal query equivalent to a given conjunctive query. We show that unless NP is contained in ZPP (the class of languages accepted by probabilistic Turing machines in expected polynomial time), a containment not believed to hold, there is no polynomial time algorithm that can approximate the minimal query to a factor better than $m^{1-\epsilon}$ (m is the number of subgoals in the given query) for every fixed $\epsilon > 0$.

There are close connections between query containment and the problem of answering queries using materialized views [15]. This problem has recently received considerable attention because of its numerous applications, which include speeding up query evaluation, querying heterogeneous information sources, mobile computing, and maintaining physical data independence ([15] provides references). For example, the Information Manifold system [16] represents the contents of heterogeneous information sources as views on a common set of base relations. A query Q is "solved" by a program that uses the views to obtain information from the sources.

We consider in this paper the problem of finding an equivalent rewriting of a conjunctive query Q using a set of views $\mathcal{V}$ defined by conjunctive queries, when Q does not use repeated predicates, and show how our algorithms for query containment can be modified for this problem. A restricted variant of this problem, where neither Q nor the views in $\mathcal{V}$ use repeated predicates, is known to be NP-complete [15].

This paper is organized as follows. Section 2 contains basic definitions and ideas related to the problems we consider. Section 3 introduces acyclicity, query width, and treewidth, and describes how these concepts relate to one another. In Section 4 we present our algorithms for query containment, and in Section 5 we prove the hardness of minimizing conjunctive queries. In Section 6 we modify the algorithms for query containment to obtain algorithms for answering queries using views. Section 7 describes related work, and Section 8 concludes by describing some open problems.

2 Preliminaries

We assume a fixed set of predicates, called the *database predicates*, over which queries are posed and views are defined. All queries in this paper are *conjunctive queries*, defined in the conventional manner [21]. We say query Q_1 is *contained* in query Q_2, denoted by $Q_1 \subseteq Q_2$, if for every state of the relations corresponding to the database predicates, the relation corresponding to the head of Q_1 is a subset of the relation corresponding to the head of Q_2. We say that Q_1 is *equivalent* to Q_2, denoted by $Q_1 \equiv Q_2$, if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

A *containment mapping* from Q_1 to Q_2 is a mapping from the variables of Q_1 to the variables and constants of Q_2 that maps each subgoal of Q_1 to a subgoal of Q_2 and also maps the head of Q_1 to the head of Q_2. For conjunctive queries, $Q_1 \subseteq Q_2$ if and only if there is a containment mapping from Q_2 to Q_1. The problems of testing whether $Q_1 \subseteq Q_2$ and $Q_1 \equiv Q_2$ are both NP-complete [7].

A *view* is a conjunctive query with a unique head predicate. Let Q be a query and $\mathcal{V}$ a set of views over the same set of database predicates. Let Q' be a query over the view predicates in $\mathcal{V}$. We extend the notions of containment and equivalence in the natural manner and make statements such as $Q \subseteq Q'$ and $Q \equiv Q'$. We call Q' an *equivalent rewriting* of Q using $\mathcal{V}$ if $Q' \equiv Q$.

Example 1 Suppose we have relations $part(Pname, Type)$, $supp(Sname, Saddr)$, $cust(Cname, Caddr)$, and $sales(Part, Supplier, Customer)$. Query Q asks for the types of parts bought by customers who have the same address as some supplier. Query Q' asks for types of parts sold by suppliers such that a customer at the same address buys parts of the same type.

$$
\begin{array}{lll}
Q: \; q(T) & :- & sales(P,S,C) \;\&\; part(P,T) \;\&cust(C,A) \;\&\; supp(S',A) \\
Q': \; q(T) & :- & sales(P,S,C) \;\&\; part(P,T) \;\&\; part(P',T) \;\& \\
 & & sales(P',S',C') \;\&\; cust(C,A) \;\&\; supp(S',A)
\end{array}
$$

In Section 4, we present a polynomial-time algorithm to verify that $Q' \subseteq Q$.

Suppose we have the materialized views V_1, V_2, and V_3 shown below. View V_1 relates customers with the types of parts they buy, V_2 gives the address of each customer who buys some part, and view V_3 gives the address of each supplier.

$$
\begin{array}{lll}
V_1: \; v_1(C_1,T_1) & :- & sales(P_1,S_1,C_1) \;\&\; cust(C_1,A_1) \;\&\; part(P_1,T_1) \\
V_2: \; v_2(C_2,A_2) & :- & sales(P_2,S_2,C_2) \;\&\; cust(C_2,A_2) \\
V_3: \; v_3(S_3,A_3) & :- & supp(S_3,A_3)
\end{array}
$$

The query Q can be equivalently rewritten using the views as follows:

$$C : q(T) \quad :- \quad v_1(C,T) \; \& \; v_2(C,A) \; \& \; v_3(S',A)$$

We can save a join by using the materialized views to answer the query. $\square$

To test whether C is an equivalent rewriting of Q, we construct the *expansion* E of C by replacing each view predicate in the body of C by its definition, using different local variables for the expansion of each view predicate. It is easily seen that $C' \equiv E$. Since Q and E are defined over the same sets of database relations, we can use containment mappings between them to test their equivalence and containment, and hence the equivalence and containment of Q and C. For example, the expansion of C in Example 1 is

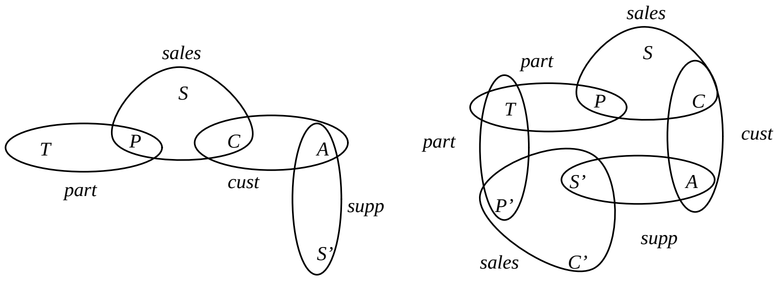
$$\begin{array}{llll} E: & q(T) & :- & sales(P_1,S_1,C) \; \& \; cust(C,A_1) \; \& \; part(P_1,T) \; \& \\ & & & sales(P_2,S_2,C) \; \& \; cust(C,A) \; \& \; supp(S',A) \end{array}$$

It can easily be verified that $E \equiv Q$, and so $C \equiv Q$.

The problem of finding an equivalent rewriting of a query Q using a set of views $\mathcal{V}$ was shown to be NP-complete by Levy et al. [15]. They show that the problem remains NP-complete even when the query and the views contain no repeated predicates. In this paper we present a polynomial-time algorithm for the equivalent rewriting problem, provided the query satisfies certain conditions to be defined in Section 3.

We now define some terminology. We use *argument* to mean either a variable or a constant that appears in a query, and *query term* to mean either an argument or a query subgoal. A *variable mapping* is a function that maps a set of variables to a set of arguments. A set of variable mappings $\phi_1, \ldots, \phi_n$, whose domains are different but perhaps overlapping, are said to be *consistent* if there do not exist variable A and integers i and j such that $\phi_i(A) \neq \phi_j(A)$. If $\phi_1, \ldots, \phi_n$ are consistent, we define their *union mapping,* ϕ, to be the variable mapping whose domain is the union of the domains of $\phi_1, \ldots, \phi_n$, and $\phi(A) = B$ if there exists some i, $1 \leq i \leq n$ such that $\phi_i(A) = B$. Variable mappings are defined to be the identity mapping on predicate symbols and constants, and so we can apply a variable mapping to any query term with the obvious meaning. If $\bar{A}$ is a tuple of arguments in the domain of a variable mapping ϕ, then $\phi(\bar{A})$ is a tuple over the set of attributes $\bar{A}$, where the value of each attribute is its image under ϕ.

A *partial mapping* ϕ from query Q to Q' is a variable mapping that maps some subset of the variables of Q to variables of Q', such that if X_i is the ith head argument of Q and $\phi(X_i) = Y_i$, then Y_i is the ith head argument of Q'. A *containment mapping* from Q to Q' is therefore a partial mapping whose domain is the set of all variables of Q.



(a) Hypergraph for Q (b) Hypergraph for Q'

Figure 1: Hypergraphs for the queries in Example 1.

3 Acyclicity, Query Width, and Treewidth

3.1 Acyclic Queries

It is often profitable to represent a conjunctive query as a hypergraph. The nodes of the hypergraph are the constants and variables in the query. There is one hyperedge corresponding to each query subgoal that includes the variables and constants occurring as arguments in that subgoal. Figure 1 shows the hypergraphs corresponding to queries Q and Q' of Example 1.

In the rest of this paper we restrict ourselves, for simplicity of exposition, to queries whose hypergraphs are connected. However, our results generalize in a straightforward manner to queries with disconnected hypergraphs.

Let E and F be hyperedges of hypergraph $\mathcal{G}$ such that the nodes in $E - F$ are *unique to* E; that is, they appear in no other hyperedge of $\mathcal{G}$. Then we call E an *ear*, the removal of E from $\mathcal{G}$ *ear removal,* and say that "E is removed in favor of F." The *GYO-reduction* of a hypergraph [10, 24] is obtained by removing ears until no further ear removals are possible. A hypergraph is *acyclic* if its GYO-reduction is the empty hypergraph; otherwise it is *cyclic.*

A query is cyclic (acyclic) if its hypergraph is cyclic (acyclic). If Q is an acyclic query, an *elimination tree* of Q is a rooted tree constructed as follows. Choose some sequence of ear removals for the hypergraph of Q. The tree has a node for each subgoal of Q, and E is a child of F in the tree whenever the hyperedge corresponding to E is eliminated in favor of the hyperedge corresponding to F in the chosen ear removal sequence.

Example 2 The hypergraph of query Q (Figure 1(a)) is acyclic. Figure 2 shows one possible elimination tree for this hypergraph. The hypergraph of Q' (Figure 1(b)) is cyclic, because it contains no ear and is its own GYO-reduction.

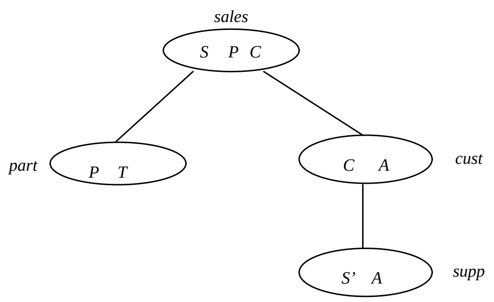


Figure 2: Elimination tree for hypergraph in Figure 1(a).

Suppose Q is an acyclic query, and T is an elimination tree for Q. An important observation that follows from the definition of the elimination tree is that for any argument X of Q, the subgoals that mention X form a connected subtree of T. It is this "connectedness property" of acyclic queries that enables a polynomial-time query containment algorithm for such queries (Section 4).

Our algorithms for acyclic queries assume the elimination tree as an input. Tarjan and Yannakakis [20] present a simple linear-time algorithm that tests whether Q is acyclic and if so, constructs an elimination tree for it.

3.2 The Width of a Query

It is natural to ask whether we can generalize the notion of query acyclicity in some way. Ideally, we would like to classify queries according to some parameter k that measures their "degree of cyclicity," such that we can design query containment algorithms whose complexity increases with k. In this section we present such a measure, which we call the *query width*.

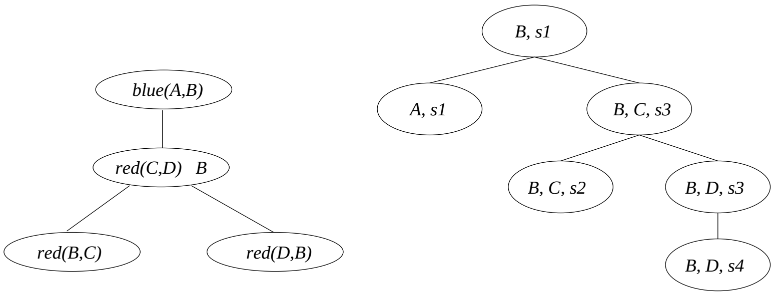
A *query decomposition* of Q is a tree $T = (I, F)$, with a set $X(i)$ of subgoals and arguments associated with each vertex $i \in I$, such that the following conditions are satisfied:

- For each subgoal s of Q, there is an $i \in I$ such that $s \in X(i)$.
- For each subgoal s of Q, the set $\{i \in I \mid s \in X(i)\}$ induces a (connected) subtree of T.
- For each argument A of Q, the set

  $$\{i \in I \mid A \in X(i)\} \cup \{i \in I \mid A \text{ appears in a subgoal } s \text{ such that } s \in X(i)\}$$

  induces a (connected) subtree of T.

The *width* of the query decomposition is $\max_{i \in I} |X(i)|$. The *query width* of Q is the minimum width over all its query decompositions.



(a) Query decomposition (b) Tree decomposition

Figure 3: A query decomposition and a tree decomposition for query Q_2 in Example 3

Example 3 Suppose *red* and *blue* are relations that represent the set of red and blue arcs in a directed graph G. Queries Q_2 and Q_3 below ask for blue arcs in subgraphs of G that satisfy certain properties.

$$\begin{array}{lll} Q_2: \ q_2(A,B) & :- & \mathit{blue}(A,B)\ \&\ \mathit{red}(B,C)\ \&\mathit{red}(C,D)\ \&\ \mathit{red}(D,B) \\ Q_3: \ q_3(X,Y) & :- & \mathit{blue}(X,Y)\ \&\ \mathit{red}(Y,Z)\ \&\mathit{red}(Z,Y)\ \&\ \mathit{red}(Z,Z) \end{array}$$

It can be verified that Q_2 is cyclic. Figure 3(a) shows a query decomposition (of width 2) of Q_2; it can be shown that the query width of Q_2 is in fact 2. Section 4 presents an efficient algorithm to test whether $Q_3 \subseteq Q_2$. □

The elimination tree of an acyclic query (with one subgoal at each node) is a query decomposition of width 1, and acyclic queries have width 1. Moreover, the query decomposition with the smallest number of nodes for a query of width 1 is also an elimination tree. The following lemma summarizes these observations.

Lemma 1 *A query is acyclic if and only if its query width is 1.*

3.3 Treewidth

Our algorithms assume that given a query Q and some constant k, we can determine efficiently whether its query width is bounded by k and if so, construct a query decomposition of width no more than k. It is open whether there is a polynomial-time algorithm for the above problem (we discuss some complexity issues at at the end of this section). However, we can obtain an upper bound on the query width by using the closely related notion of *treewidth.*

The *incidence graph* $G_Q = (V, E)$ of query Q has a vertex for each argument and for each subgoal of Q. There is an edge between an argument X and a subgoal s whenever X occurs in

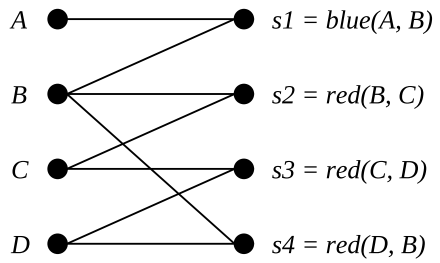


Figure 4: The incidence graph for the query Q_2.

s. Figure 4 shows the incidence graph of query Q_2 from Example 1, where we use s_1, s_2, s_3, s_4 to denote the subgoals of Q_2 in order from left to right.

A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (I, F)$, with a set $X(i) \subseteq V$ associated with each vertex $i \in I$, such that the following conditions are satisfied:

- For each $v \in V$, there is an $i \in I$ with $v \in X(i)$.
- For all edges $(v, w) \in E$, there is an $i \in I$ with $v, w \in X(i)$.
- For each $v \in V$, the set $\{i \in I \mid v \in X(i)\}$ induces a (connected) subtree of T.

The *width* of the tree decomposition is $\max_{i \in I} |X(i)| - 1$. The *treewidth* of G is the minimum width over all its tree decompositions. [1] Figure 3(b) shows a tree decomposition (width 2) of the incidence of graph in Figure 4.

The *treewidth of a query* is the treewidth of its incidence graph. We can show easily that every tree decomposition of the incidence graph of Q is also a query decomposition of Q. (For example, Figure 3(b) is another query decomposition for query Q_2.) Therefore, the query width of a query Q is certainly not more than one greater than its treewidth (due to the -1 in the definition of treewidth). In fact, we can show the following result, where $tw(Q)$ is the treewidth of Q and $qw(Q)$ is the query width of Q.

Lemma 2 *For any query Q, $tw(Q)/a \leq qw(Q) \leq tw(Q) + 1$, where a is the maximum predicate arity in Q.*

Proof: Since every tree decomposition is a query decomposition, $qw(Q) \leq tw(Q) + 1$. For any query Q, we can convert a query decomposition into a tree decomposition as follows. For each node $i \in I$ of the query decomposition (I, F), we add the set of variables occurring in all the predicates of $X(i)$. It can be verified that this transformation results in a valid tree decomposition for Q.

[1] The -1 in the definition of width ensures that trees have treewidth 1.

Since the number of variables added for each predicate is at most the arity of that predicate, the bound on the treewidth follows. $\square$

We assume that all predicate arities are bounded by some constant in this paper, and so Lemma 2 implies that the treewidth approximates the query width to within a constant factor. Lemma 2 is useful because treewidth is an extensively studied concept in graph theory. Though computing the treewidth of a graph is NP-Complete in general [3], checking a graph for small treewidth is in polynomial time. Bodlaender [6] presents a polynomial time algorithm to determine, for a given k, whether the treewidth of a graph is bounded by k. The running time of the algorithm is exponential in k but linear in the size of the graph.

Theorem 1 ([6]) *For every fixed $k \in N$, there exists a linear time algorithm that tests whether a given graph $G = (V, E)$ has treewidth at most k, and if so, outputs a tree decomposition of G with treewidth at most k that has at most $|V| - k$ nodes.*

Discussion: As mentioned earlier we do not know if the problem of computing the query width of a given query is NP-Hard. Obvious attempts at reducing the problem of computing the treewidth of a graph to computing the query width of a query do not seem to work. From a graph theoretic point of view the asymmetry between subgoals and variables in the definition of query width (only subgoals are required to be in a query decomposition) is not natural. However our definition of query width was motivated by the weaker conditions necessary for computing containment mappings (acyclic queries have query width 1 even though the treewidth of the incidence graph could be much larger). We still believe that computing query width is NP-Hard. A careful look at the proof of hardness of computing treewidth [3] might suggest possible reductions. Also, from Lemma 2 the gap between query width and treewidth is at most the maximum arity of the predicates. The natural query associated with a graph (see Section 5) has arity 2. Therefore if it is established that computing the treewidth of a graph is hard to approximate within a factor larger than 2, it would imply NP-hardness of computing query width. The best known approximation algorithm for computing treewidth has a ratio of $O(\log n)$ [5], so the above possibility is not ruled out.

In addition, even if computing query width is hard, we believe that finding a query decomposition of size k, if it exists, can be computed in polynomial time for each fixed k. Our containment algorithms are polynomial only for fixed width queries, hence establishing the complexity of this restricted case is sufficient for our purposes. We leave these issues for future research.

4 Algorithms for Query Containment

Section 4.1 presents a polynomial-time algorithm to test whether $Q' \subseteq Q$, when Q is an acyclic query (there are no restrictions on Q'). Our approach is to construct partial mappings from Q to Q' and successively merge partial mappings until we either can no longer merge mappings or have found a containment mapping from Q to Q'. The connectedness property allows us to merge partial mappings in polynomial time. In Section 4.2 we extend the algorithm to work with query decompositions of arbitrary width for Q. Given a decomposition of width k, the algorithm runs in time polynomial in n^k where n is the sum of the sizes of Q and Q'.

4.1 Containment Algorithm for Acyclic Queries

Let $T = (I, F)$ be an elimination tree for Q, and let s_i be the subgoal of Q corresponding to node $i \in I$. The algorithm maintains a relation $Map_i(\bar{A}_i)$ at each node i. The attributes $\bar{A}_i$ of the relation are the arguments of s_i (repeated arguments appear only once in $\bar{A}_i$). We use S_i to denote the set of subgoals in the subtree rooted at i. In the algorithm below, $\ltimes$ denotes the natural semijoin operator.

Algorithm AcyclicContainment

1. Initialize the relations as follows. For each partial mapping ϕ from Q to Q' that maps s_i to some subgoal of Q', the tuple $\phi(\bar{A}_i)$ is in $Map_i(\bar{A}_i)$.

2. Process tree nodes bottom-up as follows. Suppose i is a node of T all of whose children have been processed. For each child j of i:

$$Map_i(\bar{A}_i) := Map_i(\bar{A}_i) \ltimes Map_j(\bar{A}_j)$$

3. $Q' \subseteq Q$ if and only if the relation at the root of T is nonempty.

Example 4 Figure 5 shows the relations created by the algorithm when the elimination tree in Figure 2 is used to determine whether $Q' \subseteq Q$ in Example 1. The attribute names at each node give the relation schema for that node. Step 1 creates the relations in dashed boxes and Step 2 results in the relations in the solid boxes. The numbers at the nodes show the order in which they are processed in Step 2. We conclude that $Q' \subseteq Q$ since the relation at the root is nonempty after Step 2. $\Box$

Lemma 3 *Algorithm AcyclicContainment correctly determines whether $Q' \subseteq Q$.*

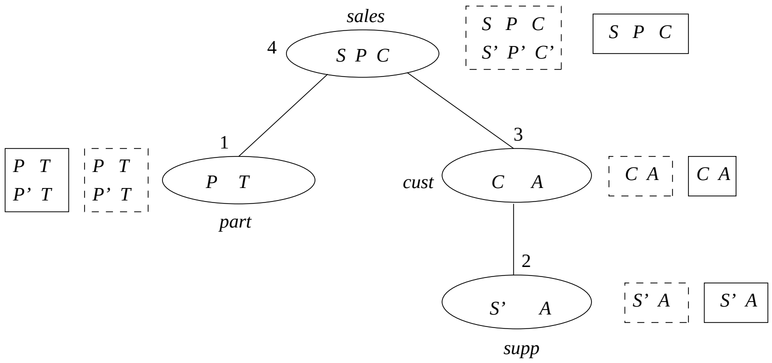


Figure 5: Running Algorithm AcyclicContainment on Example 1 using the tree in Figure 2.

Proof: We use induction on the number of nodes processed, with the following induction hypothesis: After node i is processed, tuple $t \in Map_i(\bar{A}_i)$ if and only if there is a partial mapping ϕ from Q to Q' whose domain is the set of subgoals S_i, such that $\phi(\bar{A}_i) = t$. Thus, when the root of T has been processed, the relation at the root is nonempty if and only if there is a containment mapping from Q to Q'.

The induction hypothesis holds for the leaves, because of the way the relations are initialized in Step 1. For the induction, assume that we have processed internal node i with children $j_1, \ldots, j_r$, and let $t \in Map_i$. Step 1 ensures that there is a partial mapping ψ from Q to Q' with domain s_i such that $\psi(\bar{A}_i) = t$. Step 2 assures us that for each j_k, there is a tuple $t_k \in Map_{j_k}$ that agrees with t on the attributes in $\bar{A}_i \cap \bar{A}_{j_k}$. By the induction hypothesis, there is a partial mapping ϕ_k from Q to Q' with domain S_{j_k} such that $\phi_k(\bar{A}_{j_k}) = t_k$.

The mappings ψ and ϕ_k are consistent. To see this, suppose X is a variable in the domains of both ϕ_k and ψ. By the connectedness property, $X \in \bar{A}_{j_k}$ and $X \in \bar{A}_i$. Since t and t_k agree on their common attributes, ψ and ϕ_k agree on X. Moreover, the mappings $\psi, \phi_1, \ldots, \phi_r$ are also consistent. To see this, suppose X is a variable in the domains of ϕ_k and ϕ_l. By the connectedness property, $X \in \bar{A}_{j_k}$, $X \in \bar{A}_{j_l}$, and $X \in \bar{A}_i$. Therefore, ψ, ϕ_k and ϕ_l agree on X. Let ϕ be the partial mapping from Q to Q' that is the union of $\psi, \phi_1, \ldots, \phi_r$. Then ϕ satisfies the conditions of the induction hypothesis.

Conversely, let ϕ be a partial mapping with domain S_i. Let $\phi_1, \ldots, \phi_r$ be the projection of ϕ on the sets of variables in $S_{j_1}, \ldots, S_{j_r}$, and let ψ be the projection of ϕ on the variables in s_i. By the induction hypothesis, there is a tuple $t_k \in Map_{j_k}$, $k = 1, \ldots, r$, such that $\phi_k(\bar{A}_{j_k}) = t_k$. After Step 1, there is a tuple $t \in Map_i$ such that $\psi(\bar{A}_i) = t$. Since t agrees with the tuples $t_1, \ldots, t_r$ on all common attributes, it will remain in Map_i after the sequence of joins in Step 2. $\Box$

Theorem 2 *Algorithm AcyclicContainment determines whether* $Q' \subseteq Q$ *in time* $O(N_Q N_{Q'} \log N_{Q'})$

using space $O(N_Q N_{Q'})$, where N_Q and $N_{Q'}$ are the sizes of Q and Q', respectively.

Proof: Correctness follows from Lemma 3. The space and time complexities follow from the observation that the cardinality of each relation Map_i is bounded by the number of subgoals in Q', and the number of such relations is exactly the number of subgoals in Q. The semijoins in Step 2 have to implemented as sort-merge semijoins to achieve the time complexity in the theorem. $\square$

Corollary 1 *Given an acyclic query Q, there is an algorithm to minimize Q in time $O(N_Q^3 \log N_Q)$ using space $O(N_Q^2)$, where N_Q is the size of Q.*

Proof: Obtain Q' from Q by deleting an arbitrary subgoal s_i. If $Q' \subseteq Q$, s_i is redundant; recursively minimize Q'. Otherwise, s_i is in the minimal equivalent query. We need at most N_Q invocations of the containment algorithm to minimize Q. $\square$

4.2 Generalizing the Algorithm

We now generalize Algorithm AcyclicContainment to test whether $Q' \subseteq Q$, where we are given a query decomposition $T = (I, F)$ of width k for Q. Let $X(i)$ be the set of terms associated with node i of T. As before, we associate a relation $Map_i(\bar{A}_i)$ with node i, where $\bar{A}_i$ is constructed as follows:

1. For each argument $A \in X(i)$, $A \in \bar{A}_i$.

2. For each subgoal $s \in X(i)$, $s \in \bar{A}_i$.

3. For each subgoal $s \in X(i)$, and each argument A that occurs in s, $A \in \bar{A}_i$.

We call attributes of type 1 and 2 *independent attributes* and attributes of type 3 *dependent attributes* (the reason for the nomenclature will become apparent later). Let $S(i)$ denote the set of terms associated with the nodes in the subtree of T rooted at i. The algorithm for query containment is now identical to Algorithm AcyclicContainment except for the initialization step.

Algorithm QueryContainment

1. (Initialize.) For each partial mapping ϕ from Q to Q' that maps all the terms in $X(i)$, include the tuple $\phi(\bar{A}_i)$ in Map_i.

2. (Propagate bottom-up.) Process tree nodes bottom-up as follows. Suppose i is a node of T all of whose children have been processed. For each child j of i:

$$Map_i(\bar{A}_i) := Map_i(\bar{A}_i) \ltimes Map_j(\bar{A}_j)$$

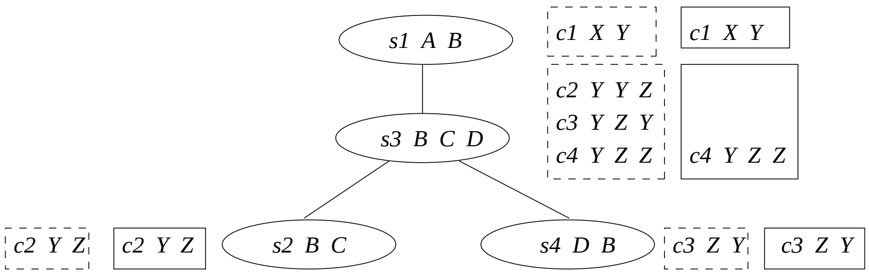


Figure 6: Running Algorithm QueryContainment

3. $Q' \subseteq Q$ if and only if the relation at the root of T is nonempty.

Example 5 Figure 6 shows the relations created by the algorithm when testing whether $Q_3 \subseteq Q_2$ in Example 3. We use the query decomposition in Figure 3(a) for Q_2. (We could have used the decomposition in Figure 3(b) as well.) In the figure $s_1, \ldots, s_4$ are the subgoals of Q_2 from left to right and $c_1, \ldots, c_4$ are the subgoals of Q_3 from left to right. The relation schema is show at each node, with independent attributes followed by dependent attributes. Step 1 creates the relations in the dashed boxes, and Step 2 results in the relations in the solid boxes. We conclude the $Q_3 \subseteq Q_2$ since the relation at the root is nonempty after Step 2. $\square$

Theorem 3 *Given a query decomposition of width k for Q, algorithm QueryContainment determines whether $Q' \subseteq Q$ in time $O(kN_Q N_{Q'}^k \log N_{Q'})$ using space $O(N_Q N_{Q'}^k)$, where N_Q and $N_{Q'}$ are the sizes of Q and Q', respectively.*

Proof: The proof of correctness is similar to that of Lemma 3. To obtain the time and space complexities, we observe that in any tuple $t \in Map_i$, the values of the independent attributes determine those of the dependent attributes (i.e., when we map a subgoal, we also map all its arguments). The number of independent attributes of each relation Map_i is bounded by the query width k of Q. Therefore, the size of Map_i is bounded by $N_{Q'}^k$, and the number of such relations is $O(N_Q)$. Using sort-merge semijoins in Step 2 gives the desired result. $\square$

Corollary 2 *Given a query decomposition of width k for Q, there is an algorithm to minimize Q in time $O(kN_Q^{k+2} \log N_Q)$ using space $O(N_Q^{k+1})$, where N_Q is the size of Q.*

5 Approximating the Minimal Equivalent Conjunctive Query

Chandra and Merlin [7] used a reduction from graph coloring to show that conjunctive query containment and minimization are NP-Complete. It is natural to view the minimization problem from an optimization point of view and consider approximation algorithms for it. In particular, we consider the following problem: given a conjunctive query Q, find a conjunctive query with the smallest number of subgoals such that Q' is equivalent to Q. It follows from the results of [7] that a *minimal* equivalent query (that cannot be further minimized) is also a *minimum* equivalent query and also that there exists a minimum equivalent query that is isomorphic to a subset of the the subgoals of Q. Hence we do not distinguish between the two notions.

We show that the above problem is hard to approximate to within a factor of $m^{1-\epsilon}$ for every $\epsilon > 0$ where m is the number of subgoals in the query. We will establish this by showing that the reduction of Chandra and Merlin from graph coloring is in fact approximation preserving, and then using the recent result of Feige and Kilian [9] on the hardness of coloring (based on a long and remarkable line of work, most notably the result of Håstad [11] on hardness of approximating clique).

Definition 1 *Given a undirected graph $G = (V, E)$, we define the conjunctive query Q_G corresponding to G as the following query.*

$$Q_G : q() : - \bigwedge_{(u,v)\in E} R(u, v) \wedge R(v, u)$$

Each edge $(u, v) \in E$ is present as both $R(u, v)$ and $R(v, u)$.

A graph is $G = (V, E)$ is p-colorable if there is a mapping h from V to $\{1, 2, \ldots, p\}$ such that for every edge $(u, v) \in E$, $h(u) \neq h(v)$. The chromatic number of G denoted by $\chi(G)$ is the smallest p such that G is p-colorable. Given two graphs G and H with *disjoint* vertex sets, we use the notation $G + H$ to denote the graph with the vertex set $V_G \cup V_H$ and the edge set $E_G \cup E_H$. We prove the following lemma for the sake of completeness.

Lemma 4 *[7] A graph G is p-colorable if and only if $Q_1 = Q_{K_p}$ is equivalent to $Q_2 = Q_{G+K_p}$ where K_p is the complete graph on p vertices.*

Proof: It is clear that $Q_2 \subseteq Q_1$ for all G and p. Therefore it is sufficient to show that G is p-colorable if and only if $Q_1 \subseteq Q_2$. Let the vertices of G be labeled $v_1, v_2, \ldots, v_n$ and the vertices of K_p be labeled $u_1, \ldots, u_p$. Suppose G is p-colorable. Then there is a mapping h from V_G to $\{1, 2, \ldots, p\}$ such that for every edge $(v_i, v_j) \in E_G$, $h(v_i) \neq h(v_j)$. We will use h to define a containment mapping g from Q_2 to Q_1. For each vertex v_i define $g(v_i) = u_{h(v_i)}$ and to be the identity on the vertices of K_p. From the validity of the coloring of h it follows that g is indeed a containment mapping. The other direction is also similar. If there is containment mapping g from Q_2 to Q_1, then each vertex v_i is mapped to some vertex u_j. We assign the color j to every vertex mapped to u_j and this yields a valid coloring of G using p colors. $\square$

The following lemma follows from elementary graph theory.

Lemma 5 *For any graph $G = (V, E)$, $|E| \geq \chi(G)(\chi(G)-1)/2$ where $\chi(G)$ is the chromatic number of G.*

Lemma 6 *Let Q' be any conjunctive query equivalent to Q_G where G is a graph with n vertices. Let m be the number of subgoals in Q'. Then $m \geq \chi(G)(\chi(G)-1)$ and $\sqrt{m+n} \geq \chi(G)$.*

Proof: It is an easy observation that $Q' = Q_{G'}$ for some graph G'. From the proof of Lemma 4 it can be inferred that for any graph H, $Q_{K_p} \subseteq Q_H$ if and only if $p \geq \chi(H)$. It follows that $Q_{K_{\chi(G')}} \subseteq Q_{G'}$. Since $Q_{G'}$ is equivalent to Q_G we get $Q_{K_{\chi(G')}} \subseteq Q_{G'} \subseteq Q_G$. But this implies that $\chi(G') \geq \chi(G)$. In fact we can infer that $\chi(G') = \chi(G)$ using the symmetry of equivalence. From Lemma 5 the number of edges in G' is at least $\chi(G)(\chi(G)-1)/2$. Therefore the number of subgoals in Q', m, is at least $\chi(G)(\chi(G)-1)$ (since each edge leads to two subgoals). Since $\chi(G) \leq n$ a simple manipulation yields the inequality $\sqrt{m+n} \geq \chi(G)$. □

Theorem 4 *Unless $NP = ZPP$[2], there is no polynomial time approximation algorithm for the problem of minimizing conjunctive queries with an approximation guarantee better than $m^{1-\epsilon}$ for every $\epsilon > 0$, where m is the number of subgoals in the query.*

Proof: Feige and Kilian [9] show that unless $NP = ZPP$, there is no polynomial time algorithm that can approximate the chromatic number of graph to a factor better than $n^{1-\epsilon}$ for every $\epsilon > 0$, where n is the number of nodes in the graph. Let $\mathcal{A}$ be any polynomial time approximation algorithm for minimizing conjunctive queries whose approximation guarantee is bounded by $m^{1-\delta}$ for some fixed $0 < \delta < 1/2$. We will construct an algorithm $\mathcal{A}'$ for chromatic number that has an approximation guarantee of $n^{1-\delta/2}$. Given a graph G, $\mathcal{A}'$ gives the query $Q = Q_{G+K_p}$ to $\mathcal{A}$ where $p = n^{\delta/2}$. $\mathcal{A}'$ returns $\min(\sqrt{m'+n}, n)$ as the estimate for $\chi(G)$ where m' is the number of subgoals in the query returned by $\mathcal{A}$. From Lemma 6 it follows that we never return an estimate that is less than the chromatic number of G.

The number of subgoals in Q, m, is at most $n(n-1) < n^2$. Let t be the number of subgoals in a minimal query equivalent to Q. Observe that $\chi(G+K_p) = \max(p, \chi(G))$. From Lemma 4 it follows that if $\chi(G) \leq p$, then $t \leq p(p-1) < p^2 = n^\delta$. If $\chi(G) > p$, from Lemma 6, $t \geq \chi(G)(\chi(G)-1)$. We now show that $\min(\sqrt{m'+n}, n)$ provides the claimed approximation to $\chi(G)$ by analyzing the above two cases separately.

[2] ZPP is the class of languages that can be decided by a randomized Turing machine in expected polynomial time. Weaker hardness results for coloring are known under the stronger $P \neq NP$ assumption.

- $\chi(G) > p$. As $\min(\sqrt{m'+n}, n) \leq n$, the approximation ratio in this case is bounded by $n/\chi(G) \leq n/p \leq n^{1-\delta/2}$.
- $\chi(G) \leq p$. As claimed above, t the number of subgoals in a minimal query for Q satisfies $t < p^2 = n^{\delta}$. Since $\mathcal{A}$ provides a $m^{1-\delta}$ approximation, $m' \leq m^{1-\delta} \cdot t \leq n^{2-2\delta} \cdot n^{\delta} \leq n^{2-\delta}$. Therefore $\sqrt{m'+n} = O(n^{1-\delta/2})$ and since $\chi(G) \geq 1$, the approximation guarantee in this case is $O(n^{1-\delta/2})$.

Thus a $m^{1-\delta}$ approximation to conjunctive query minimization provides a $n^{1-\delta/2}$ approximation for chromatic number. The hardness of approximation result of [9] implies that this is not possible unless $NP = ZPP$ for any constant $\delta > 0$. $\square$

6 Algorithms for Answering Queries using Views

Let $p_1, \ldots, p_n$ be distinct predicates, and let Q be the query

$$Q : q(\bar{X}) \ :- \ p_1(\bar{X_1}) \ \& \ \ldots \ \& \ p_n(\bar{X_n})$$

We wish to determine whether there is an equivalent rewriting of Q using an arbitrary set of views $\mathcal{V}$.

Call $V \in \mathcal{V}$ an *interesting view* if there is a variable mapping ϕ that maps the subgoals of V into subgoals of Q (ϕ need not map the head of V to the head of Q). The mapping ϕ is unique for a given interesting view V: since Q has no repeated predicates, there is a unique destination for each subgoal of V. Let $V_1, \ldots, V_m$ be the interesting views in $\mathcal{V}$ with associated mappings $\phi_1, \ldots, \phi_m$, and let $v_i(\bar{Y_i})$ be the head of view V_i. Then the *canonical rewriting of* Q *using* $\mathcal{V}$ is

$$C : q(\bar{X}) \ :- \ v_1(\phi_1(\bar{Y_1})) \ \& \ \ldots \ \& \ v_m(\phi_m(\bar{Y_m}))$$

Example 6 In Example 1, all three views V_1, V_2, and V_3 are interesting for query Q. The rewriting C is the canonical rewriting in this case. $\square$

Lemma 7 *There is an equivalent rewriting of Q using $\mathcal{V}$ if and only if $C' \subseteq Q$, where C' is the expansion of the canonical rewriting C.*

Proof: (If.) Suppose $C' \subseteq Q$. We will show that $Q \subseteq C'$; it follows that $Q \equiv C'$ and C is an equivalent rewriting of Q using $\mathcal{V}$. Let S_i denote the set of subgoals contributed to C' by the expansion of $v_i(\phi_i(\bar{Y_i}))$. There is a unique variable mapping ψ_i that maps the subgoals in S_i to subgoals of Q, corresponding to the unique mapping ϕ_i from V_i to Q. Moreover ψ_i is the identity on the variables in $\phi_i(\bar{Y_i})$. For any i and j, the only variables common to the subgoals in S_i and S_j occur in $\phi_i(\bar{Y_i})$ and $\phi_j(\bar{Y_j})$. Since ψ_i and ψ_j are both the identity mapping on these variables, ψ_i and ψ_j are consistent. It follows that $\psi_1, \ldots, \psi_m$ is a consistent set of mappings. Let ψ be the union mapping of $\psi_1, \ldots, \psi_m$. Then ψ is a containment mapping from C' to Q, showing that $Q \subseteq C'$.

(Only if.) Conversely, suppose the following is an equivalent rewriting of Q using $\mathcal{V}$:

$$R : q(\bar{X}) \ :- \ u_1(\bar{Z_1}) \ \& \ \ldots \ \& \ u_l(\bar{Z_l})$$

Let R' be the expansion of R. Since $Q \equiv R'$, there is a containment mapping τ from Q to R' and a containment mapping ρ from R' to Q. The projection of ρ on to the expansion of u_i maps the subgoals of the corresponding view to subgoals to Q, and so u_i is an interesting view for $i = 1, \ldots, l$. Let us assume without loss of generality that u_i is identical to v_i, $i = 1, \ldots, l$.

Consider the query T defined as follows:

$$T : q(\bar{X}) \ :- \ v_1(\rho(\bar{Z_1})) \ \& \ \ldots \ \& \ v_l(\rho(\bar{Z_l}))$$

Let T' be the expansion of T. We have constructed T from R in a such a manner as to ensure that T is an equivalent rewriting of Q and $T' \equiv Q$. Since there is a unique mapping ϕ_i from each interesting view V_i to Q, it must be the case that $\phi_i(\bar{Y_i}) = \rho(\bar{Z_i})$. Thus, each subgoal in T is also a subgoal of the canonical rewriting C, and therefore $C \subseteq T$ and $C' \subseteq T'$. Since $T' \equiv Q$, we have shown that $C' \subseteq Q$. $\square$

Theorem 5 *Let Q be a query without repeated predicates and $\mathcal{V}$ a set of views. Given a query decomposition of width k for Q, there is an algorithm to test whether there is an equivalent rewriting of Q using $\mathcal{V}$ in time $O(kN_Q|\mathcal{V}|^k \log |\mathcal{V}|)$ and space $O(N_Q|\mathcal{V}|^k)$, where N_Q is the size of Q and $|\mathcal{V}|$ is the size of $\mathcal{V}$.*

Proof: Follows from Lemma 7 and Theorem 3. $\square$

The size of the canonical rewriting depends on the number of interesting views and is independent of the size of Q. We can modify the query containment algorithms of Section 4 slightly to obtain a rewriting that uses no more subgoals than the query. With each tuple t in a relation at a node of the query decomposition, we must store also the names of the views whose subgoals participate in the partial mapping corresponding to t. After Step 2 of the containment algorithm, we traverse the tree top-down and choose a set of consistent tuples, one from each node of the tree. We can then drop all but the views corresponding to the chosen tuples from the canonical rewriting C to obtain a shorter rewriting D such that $D \equiv Q$.

Theorem 6 *Let Q be a query without repeated predicates and $\mathcal{V}$ a set of views, such that Q has n subgoals. Given a query decomposition of width k for Q, there is an algorithm that tests whether there is an equivalent rewriting of Q using $\mathcal{V}$, and if so, produces a rewriting of Q using $\mathcal{V}$ that has no more than n subgoals. The algorithm runs in time $O(k^2 N_Q |\mathcal{V}|^k \log |\mathcal{V}|)$ and uses space $O(N_Q|\mathcal{V}|^k)$, where N_Q is the size of Q and $|\mathcal{V}|$ is the size of $\mathcal{V}$.*

7 Related Work

Aho et al. [1, 2] gave polynomial-time minimization and equivalence algorithms for conjunctive queries corresponding to *simple tableaux.* Their results were extended by Johnson and Klug [12] to the class of *fanout-free queries.* Biskup et al. [4] extending the ideas in [1, 2] and [12] considered a class of typed fanout queries that are obtained from simple tableaux by allowing at most one attribute to violate the simple-tableau property. They gave quadratic time algorithms for minimization. The classes of fanout-free queries and queries with simple tableaux are incomparable to the classes of queries considered in this paper. The algorithms of Aho et al. , Johnson and Klug, and Biskup et al. also differ from ours in that they cannot be generalized to test query containment instead of query equivalence. Recently, Qian [18] independently showed that acyclic queries admit polynomial time algorithms for containment and minimization. Our work treats acyclic queries as a special case of queries with width 1, and so Qian's algorithm for query containment falls out as a special case.

Yang and Larson [14, 22] considered the problem of finding rewritings for SPJ queries using SPJ views. In their analysis, they considered what amounts to 1-1 mappings from the views to the query, and therefore their algorithm can miss some rewritings. The problem of finding equivalent rewritings was studied formally by Levy et al. [15], who showed the problem to be NP-complete. Rajaraman et al. [19] extended the results of Levy et al. to queries and views with *binding patterns.* Chaudhuri et al. [8] considered the problem of finding rewritings for SPJ queries and views, when the queries and views use bag semantics instead of the usual set semantics. They also suggest a way to extend a traditional relational query processor to choose between different rewritings based on their costs, a question we do not address in this paper. Qian [18] presents a polynomial-time algorithm that, given an acyclic query Q and a set of views $\mathcal{V}$, determines whether there is a rewriting using $\mathcal{V}$ that is contained in (but not necessarily equivalent to) Q. Levy et al. [17] study the problem of finding an equivalent rewriting when the set of views is possibly infinite, albeit encoded in some finite fashion. The Information Manifold system [16] implements heuristics that speed up the search for a rewriting of a query by eliminating irrelevant views.

Acyclic queries and acyclic database schemes have been studied extensively because their structural properties permit efficient algorithms. Several algorithms for acyclic database schemes were given by Yannakakis [23]. Our query containment algorithm for acyclic queries is closely related to the one in [23] for computing projections of acyclic joins. In fact, our results can be viewed as a combination of:

- Reducing testing query containment to the problem of testing the non-emptiness of a join; and
- Introducing a wider class of joins with an efficient test for emptiness.

In the case where the join is acyclic, we can directly apply the algorithm of [23] to perform the emptiness test in polynomial time.

Treewidth is extensively studied in the graph-theoretic literature, and several intractable problems admit polynomial-time algorithms on graphs of constant treewidth [6].

8 Open Problems

Our work raises some interesting open problems. While we obtained bounds on the query width of Q based on the treewidth of its incidence graph, it would be useful to have an efficient algorithm that produces query decompositions of small query width, analogous to the algorithm of Bodlaender [6] for decompositions of small treewidth. The connectedness property of acyclic queries leads to several efficient algorithms [23]; it may be possible to generalize many of these algorithms to queries of small query width. Finally, we would like to extend our results to queries with binding patterns [19] and queries with built-in predicates.

Acknowledgments

The inspiration for considering the treewidth of the incidence graph came from the work of Khanna and Motwani [13]. We also thank Jeff Ullman for comments on earlier versions of this paper.

References

[1] A.V. Aho, Y. Sagiv, and J.D. Ullman. Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, 4(4):435–454, December 1979.

[2] A.V. Aho, Y. Sagiv, and J.D. Ullman. Equivalence of relational expressions. *SIAM Journal on Computing*, 8(2):218–246, May 1979.

[3] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8:277–284, 1987.

[4] J. Biskup, P. Dublish, and Y. Sagiv. Optimization of a subclass of conjunctive queries. *Acta Informatica*, 32(1):1–26, 1995.

[5] H. Bodlaender, J. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, March 1995.

[6] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 226–234, 1993.

[7] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[8] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 190–200, 1995.

[9] U. Feige and J. Kilian. Zero knowledge and the chromatic number. In *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity*, pages 278–287, 1996.

[10] M.H. Graham. On the universal relation. Technical report, University of Toronto, Ontario, Canada, 1979.

[11] J. Hastad. Clique is hard to approximate to within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 627–636, 1996.

[12] D.S. Johnson and A. Klug. Optimizing conjunctive queries that contain untyped variables. *SIAM Journal on Computing*, 12(4):616–640, November 1983.

[13] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, 1996.

[14] P.A. Larson and H.Z. Yang. Computing queries from derived relations. In *Proceedings of the Eleventh International Conference on Very Large Data Bases*, pages 259–269, 1985.

[15] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems*, pages 95–104, 1995.

[16] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, 1996.

[17] A.Y. Levy, A. Rajaraman, and J.D. Ullman. Answering queries using limited external query processors. In *Proceedings of the Fifteenth ACM Symposium on Principles of Database Systems*, pages 227–237, 1996.

[18] X. Qian. Query folding. In *Proceedings of the Twelfth International Conference on Data Engineering*, 1996.

[19] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems*, pages 105–112, 1995.

[20] R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[21] J.D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II: The New Technologies.* Computer Science Press, Rockville, MD, 1989.

[22] H.Z. Yang and P.A. Larson. Query transformation for PSJ-queries. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, pages 245–254, 1987.

[23] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases*, pages 82–94, 1981.

[24] C.T. Yu and M.Z. Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *Proceedings of IEEE COMPSAC*, pages 306–312, 1979.