Answering Conjunctive Queries with Inequalities*

Paraschos Koutris[1], Tova Milo[2], Sudeepa Roy[3], and Dan Suciu[4]

1 **University of Washington**
pkoutris@cs.washington.edu
2 **Tel Aviv University**
milo@cs.tau.ac.il
3 **University of Washington**
sudeepa@cs.washington.edu
4 **University of Washington**
suciu@cs.washington.edu

Abstract

In this parer, we study the complexity of answering conjunctive queries (CQ) with inequalities ($\neq$). In particular, we compare the complexity of the query with and without inequalities. The main contribution of our work is a novel combinatorial technique that enables the use of any Select-Project-Join query plan for a given CQ without inequalities in answering the CQ with inequalities, with an additional factor in running time that only depends on the query. To achieve this, we define a new projection operator that keeps a small representation (independent of the size of the database) of the set of input tuples that map to each tuple in the output of the projection; this representation is used to evaluate all the inequalities in the query. Second, we generalize a result by Papadimitriou-Yannakakis [18] and give an alternative algorithm based on the color-coding technique [4] to evaluate a CQ with inequalities by using an algorithm for the CQ without inequalities. Third, we investigate the structure of the query graph, inequality graph, and the augmented query graph with inequalities, and show that even if the query and the inequality graphs have bounded treewidth, the augmented graph not only can have an unbounded treewidth but can also be NP-hard to evaluate. Further, we illustrate classes of queries and inequalities where the augmented graphs have unbounded treewidth, but the CQ with inequalities can be evaluated in poly-time. Finally, we give necessary properties and sufficient properties that allow a class of CQs to have poly-time combined complexity with respect to any inequality pattern.

1998 ACM Subject Classification H.2.4 [Systems]: Query Processing

Keywords and phrases query evaluation, conjunctive query, inequality, treewidth

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.76

1 Introduction

In this paper, we study the complexity of answering conjunctive queries (CQ) with a set of inequalities of the form $x_i \neq x_j$ between variables in the query. The complexity of answering CQs without inequalities has been extensively studied in the literature during the past three decades. Query evaluation of CQs is NP-hard in terms of *combined complexity* (both query and database are inputs), while the *data complexity* of CQs (query is fixed) is in AC_0 [1]. Yannakakis [23] showed that evaluation of acyclic CQs has polynomial-time combined

* This work has been partially funded by the NSF awards IIS-1247469 and IIS-0911036, European Research Council under the FP7, ERC grant MoDaS, agreement 291071 and the Israel Ministry of Science.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 76–93

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity. This result has been generalized later to CQs with bounded treewidth, bounded querywidth, or bounded hypertreewidth: the combined complexity remains polynomial if the width of a tree or query decomposition of the query (hyper-)graph is bounded [6, 10, 14, 9].

However, the complexity of query evaluation changes drastically once we add inequalities in the body of the query. Consider the following Boolean acyclic CQ P^k which can be solved in $O(k|D|)$ time on a database instance D:

$$P^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$$

If we add the inequalities $x_i \neq x_j$ for every $i < j$ and evaluate it on an instance where each $R_\ell, 1 \leq \ell \leq k$, corresponds to the edges in a graph with $k+1$ vertices, query evaluation becomes equivalent to asking whether the graph contains a Hamiltonian path, and therefore is NP-hard in k. Papadimitriou and Yannakakis [18] observed this fact and showed that still the problem is *fixed-parameter tractable* for acyclic CQs:

▶ **Theorem 1** ([18]). *Let q be an acyclic conjunctive query with inequalities and D be a database instance. Then, q can be evaluated in time $2^{O(k \log k)} \cdot |D| \log^2 |D|$ where k is the number of variables in q that appear in some inequality.*

The proof is based on the *color-coding* technique introduced by Alon-Yuster-Zwick in [4] that finds subgraphs in a graph. In general, answering CQs with inequalities is closely related to finding patterns in a graph, which has been extensively studied in the context of graph theory and algorithms. For example, using the idea of *representative sets*, Monien [16] showed the following: given a graph $G(V, E)$ and a vertex $s \in V$, there exists a deterministic $O(k! \cdot |E|)$ algorithm that finds all vertices v with a length-k path from s and also reports these paths (a trivial algorithm will run in time $O(|V|^k)$). Later, Alon *et al.* proposed the much simpler color-coding technique that can solve the same problem in expected time $2^{O(k)}|V|$ for undirected graphs and $2^{O(k)}|E|$ for directed graphs. These two ideas have been widely used to find other patterns in a graph, *e.g.*, for finding cycles of even length [3, 25, 4].

In the context of databases, Papadimitriou and Yannakakis [18] showed that answering acyclic CQs with comparison operators between variables ($<, \leq$ etc.) is harder than answering acyclic CQs with inequalities ($\neq$) since this problem is no longer fixed-parameter tractable. The query containment problem for CQs with comparisons and inequalities ($\neq, <, \leq$), *i.e.*, whether $Q_1 \subseteq Q_2$, has been shown to be Π_2^p-complete by van der Meyden [21]; the effect of several syntactic properties of Q_1, Q_2 on the complexity of this problem has been studied by Kolaitis *et al.* [14]. Durand and Grandjean [8] improved Theorem 1 from [18] by reducing the time complexity by a $log^2|D|$ factor. Answering queries with views in the presence of comparison operators has been studied by Afrati *et al.* [2]. Rosati [20] showed that answering CQs with inequalities is undecidable in description logic.

Our Contributions. In this paper we focus on the combined complexity of answering CQs with inequalities ($\neq$) where we explore both the structure of the query and the inequalities. Let q be a CQ with a set of variables, $\mathcal{I}$ be a set of inequalities of the form $x_i \neq x_j$, and k be the number of variables that appear in one of the inequalities in $\mathcal{I}$ ($k < |q|$). We will use $(q, \mathcal{I})$ to denote q with inequalities $\mathcal{I}$, and D to denote the database instance. We will refer to the combined complexity in $|D|, |q|, k$ by default (and not the data complexity on $|D|$) unless mentioned otherwise.

The main result in this paper says that any query plan for evaluating a CQ can be converted to a query plan for evaluating the same CQ with arbitrary inequalities, and the increase in running time is a factor that only depends on the query:

▶ **Theorem 2 (Main Theorem).** *Let q be a CQ that can be evaluated in time $T(|q|, |D|)$ using a Select-Project-Join (SPJ) query plan $\mathcal{P}_q$. Then, a query plan $\mathcal{P}_{q,\mathcal{I}}$ for $(q, \mathcal{I})$ can be obtained to evaluate $(q, \mathcal{I})$ in time $g(q, \mathcal{I}) \cdot \max(T(|q|, |D|),\ |D|)$ for a function g that is independent of the input database.* [1]

The key techniques used to prove the above theorem (Sections 3 and 4), and our other contributions in this paper (Sections 5, 6, and 7) are summarized below.

1. **(Section 3, 4)** Our main technical contribution is a new *projection* operator, called $\mathcal{H}$-projection. While the standard projection in relational algebra removes all other attributes for each tuple in the output, the new operator computes and retains a certain representation of the group of input tuples that contribute to each tuple in the output. This representation is of size independent of the database and allows the updated query plan to still correctly filter out certain tuples that do not satisfy the inequalities. In Section 3 we present the basic algorithmic components of this operator. In Section 4, we show how to apply this operator to transform the given query plan to another query plan that incorporates the added inequalities.
2. **(Section 5)** We generalize Theorem 1 to arbitrary CQs with inequalities (*i.e.*, not necessarily acyclic) by a simple application of the color-coding technique. In particular, we show (Theorem 21) that any algorithm that computes a CQ q on a database D in time $T(|q|, |D|)$ can be extended to an algorithm that can evaluate $(q, \mathcal{I})$ in time $f(k) \cdot \log(|D|) \cdot T(|q|, |D|)$. While Theorem 2 and Theorem 21 appear similar, there are several advantages of using our algorithm over the color-coding-based technique which we also discuss in Section 5.
3. **(Section 6)** The multiplicative factors dependent on the query in Theorem 1, Theorem 21, and (in the worst case) Theorem 2 are exponential in k. In Section 6 we investigate the combined structure of the queries and inequalities that allow or forbid poly-time combined complexity. We show that, even if q and $\mathcal{I}$ have a simple structure, answering $(q, \mathcal{I})$ can be NP-hard in k (Proposition 25). We also present a connection with the list coloring problem that allows us to answer certain pairings of queries with inequalities in poly-time combined complexity (Proposition 27).
4. **(Section 7)** We provide a sufficient condition for CQs, *bounded fractional vertex cover*, that ensures poly-time combined complexity when evaluated with *any set of inequalities* $\mathcal{I}$. Moreover, we show that families of CQs with unbounded integer vertex cover are NP-hard to evaluate in k (Theorem 29).

2 Preliminaries

We are given a CQ q, a set of inequalities $\mathcal{I}$, and a database instance D. The goal is to evaluate the query with inequality, denoted by $(q, \mathcal{I})$, on D. We will use $vars(q)$ to denote the variables in the body of query q and `Dom` to denote the active domain of D. The set of variables in the head of q (*i.e.*, the variables that appear in the output of q) is denoted by $head(q)$. If $head(q) = \emptyset$, q is called a *Boolean query*, while if $head(q) = vars(q)$, it is called a *full query*.

The set $\mathcal{I}$ contains inequalities of the form $x_i \neq x_j$, where $x_i, x_j \in vars(q)$ such that they belong to two distinct relational atoms in the query. We do not consider inequalities of the

[1] Some queries like $q() = R(x)S(y)$ can be evaluated in constant time whereas to evaluate the inequality constraints we need to scan the relations in D.

form $x_i \neq c$ for some constant c, or of the form $x_i \neq x_j$ where x_i, x_j only belong to the same relational atoms because these can be preprocessed by scanning the database instance and filtering out the tuples that violate these inequalities in time $O(|\mathcal{I}||D|)$. We will use k to denote the number of variables appearing in $\mathcal{I}$ ($k \leq |vars(q)| < |q|$).

Query Graph, Inequality Graph, and Augmented Graph. Given a CQ q and a set of inequalities $\mathcal{I}$, we define three undirected graphs on $vars(q)$ as the set of vertices:

The *query incidence graph* or simply the *query graph*, denoted by G^q, of a query q contains all the variables and the relational atoms in the query as vertices; an edge exists between a variable x and an atom S if and only if x appears in S.

The *inequality graph* $G^{\mathcal{I}}$ adds an edge between $x_i, x_j \in vars(q)$ if the inequality $x_i \neq x_j$ belongs to $\mathcal{I}$.

The query $(q, \mathcal{I})$ can be viewed as an augmentation of q with additional predicates, where for each inequality $x_i \neq x_j$ we add a relational atom $I_{ij}(x_i, x_j)$ to the query q, and add new relations I_{ij} to D instantiated to tuples $(a, b) \in \texttt{Dom} \times \texttt{Dom}$ such that $a \neq b$. The *augmented graph* $G^{q,\mathcal{I}}$ is the query incidence graph of this augmented query. Note that $G^{q,\mathcal{I}}$ includes the edges from G^q, and for every edge $(x_i, x_j) \in G^{\mathcal{I}}$, it includes two edges $(x_i, I_{ij}), (x_j, I_{ij})$; examples can be found in Section 6.

Treewidth and Acyclicity of a Query. We briefly review the definition of the treewidth of a graph and a query.

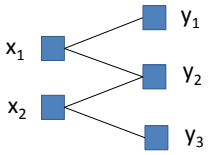
▶ **Definition 3** (Treewidth). A *tree decomposition* [19] of a graph $G(V, E)$ is a tree $T = (I, F)$, with a set $X(u) \subseteq V$ associated with each vertex $u \in I$ of the tree, such that the following conditions are satisfied:

1. For each $v \in V$, there is a $u \in I$ such that $v \in X(u)$,
2. For all edges $(v_1, v_2) \in E$, there is a $u \in I$ with $v_1, v_2 \in X(u)$,
3. For each $v \in V$, the set $\{u \in I \ : \ v \in X(u)\}$ induces a connected subtree of T.

The width of the tree decomposition $T = (I, F)$ is $\max_{u \in I} |X(u)| - 1$. The *treewidth* of G is the width of the tree decomposition of G having the minimum width.

Chekuri and Rajaraman defined the *treewidth of a query* q as the treewidth of the query incidence graph G^q [6]. A query can be viewed as a *hypergraph* where every hyperedge corresponds to an atom in the query and comprises the variables as vertices that belong to the relational atom. The *GYO-reduction* [11, 24] of a query repeatedly removes *ears* from the query hypergraph (hyperedges having at least one variable that does not belong to any other hyperedge) until no further ears exist. A query is *acyclic* if its GYO-reduction is the empty hypergraph, otherwise it is cyclic. For example, the query $P^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$ is acyclic, whereas the query $C^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_1)$ is cyclic.

There is another notion of width of a query called *querywidth* qw defined in terms of *query decomposition* such that the decomposition tree has relational atoms from the query instead of variables [6]; The relation between the querywidth qw and treewidth tw of a query is given by the inequality $tw/a \leq qw \leq tw + 1$, where a is the maximum arity of an atom in q. A query is acyclic if and only if its querywidth is 1; the treewidth of an acyclic query can be > 1 [6]. The notion of *hypertreewidth* has been defined by Gottlob *et al.* in [10]. A query can be evaluated in poly-time combined complexity if its treewidth, querywidth, or hypertreewidth is bounded [23, 6, 10, 14, 9].



(a) The bipartite graph $\mathcal{H}_0$

$$R = \{(1,1), (1,2), (1,4), (1,8), (2,1), (2,2), (2,3), (2,4), (3,2), (5,2), (10,2)\}$$

(b) The instance of $R(x_1, x_2)$

Figure 1 The running example (Example 5) for Section 3.

3 Main Techniques

In this section, we present the main techniques used to prove Theorem 2 with the help of a simple query q_2 that computes the cross product of two relations and projects onto the empty set. In particular, we consider the query $(q_2, \mathcal{I})$ with an arbitrary set of inequalities $\mathcal{I}$, where $q_2(\ ) = R(x_1, \ldots, x_m), S(y_1, \ldots, y_\ell)$. A naïve way to evaluate the query $(q_2, \mathcal{I})$ is to iterate over all pairs of tuples from R and S, and check if any such pair satisfies the inequalities in $\mathcal{I}$. This algorithm runs in time $O(m\ell|R||S|)$. We will show instead how to evaluate $(q_2, \mathcal{I})$ in time $f(q_2, \mathcal{I})(|R| + |S|)$ for some function f that is independent of the relations R and S.

The key idea is to compress the information that we need from R to evaluate the inequalities by computing a representation R' of R of such that the size of R' only depends on $\mathcal{I}$ and not on R. Further, we must be able to compute R' in time $O(f'(\mathcal{I})|R|)$. Then, instead of iterating over the pairs of tuples from R, S, we can iterate over the pairs from R' and S, which can be done in time $f''(q_2, \mathcal{I})|S|$. The challenge is to show that such a representation R' exists and that we can compute it efficiently.

We now formalize the above intuition. Let $X = \{x_1, \cdots, x_m\}$, $Y = \{y_1, \cdots, y_\ell\}$. Let $\mathcal{H} = G^{\mathcal{I}}$ denote the inequality graph; since q_2 has only two relations, $\mathcal{H}$ is a bipartite graph on X and Y. If a tuple t from S satisfies the inequalities in $\mathcal{I}$ when paired with at least one tuple in R, we say that t is $\mathcal{H}$-accepted by R, and it contributes to the answer of $(q_2, \mathcal{I})$. For a variable x_i and a tuple t, let $t[x_i]$ denotes the value of the attribute of t that corresponds to variable x_i.

▶ **Definition 4** ($\mathcal{H}$-accepted Tuples)**.** Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph. We say that a tuple t over Y is *$\mathcal{H}$-accepted* by a relation R if there exists some tuple $t_R \in R$ such that for every $(x_i, y_j) \in E$, we have $t_R[x_i] \neq t[y_j]$.

Notice that $(q_2, \mathcal{I})$ is true if and only if there exists a tuple $t_S \in S$ that is $\mathcal{H}$-accepted by R.

▶ **Example 5** (Running Example)**.** Let us define $\mathcal{H}_0 = (X, Y, E)$ with $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$ and $E = \{(x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_3)\}$ (see Figure 1(a)) and consider the instance for R as depicted in Figure 1(b). This setting will be used as our running example.

Observe that the tuple $t = (2, 1, 3)$ is $\mathcal{H}_0$-accepted by R. Indeed consider the tuple $t' = (3, 2)$ in R: it is easy to check that all inequalities are satisfied by t, t'. In contrast, the tuple $(2, 1, 2)$ is not $\mathcal{H}_0$-accepted by R.

▶ **Definition 6** ($\mathcal{H}$-Equivalence)**.** Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph. Two relations R_1, R_2 of arity $m = |X|$ are *$\mathcal{H}$-equivalent* if for any tuple t of arity $\ell = |Y|$, the tuple t is $\mathcal{H}$-accepted by R_1 if and only if t is $\mathcal{H}$-accepted by R_2.

$\mathcal{H}$-equivalent relations form an equivalence class comprising instances of the same arity m. The main result in this section shows that for a given R, an $\mathcal{H}$-equivalent instance $R' \subseteq R$ of size independent of R can be efficiently constructed.

▶ **Theorem 7.** *Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph ($|Y| = \ell$) and R be a relation of arity $m = |X|$. Let $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$, where $d_{\mathcal{H}}(v)$ is the degree of a vertex v in $\mathcal{H}$. There exists an instance $R' \subseteq R$ such that:*

1. *R' is $\mathcal{H}$-equivalent with R*
2. *$|R'| \leq e \cdot \phi(\mathcal{H})$*
3. *R' can be computed in time $O(\phi(\mathcal{H})|R|)$.*

To describe how the algorithm that constructs R' works, we need to introduce another notion that describes the tuples of arity ℓ that are *not* $\mathcal{H}$-accepted by R. Let $\perp$ be a value that does not appear in the active domain $\texttt{Dom}$.

▶ **Definition 8** ($\mathcal{H}$-Forbidden Tuples)**.** Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph and R be a relation of arity $m = |X|$. A tuple t over Y with values in $\texttt{Dom} \cup \{\perp\}$ is *$\mathcal{H}$-forbidden* for R if for any tuple $t_R \in R$ there exist $y_j \in Y$ and $(x_i, y_j) \in E$ such that $t[y_j] = t_R[x_i]$.

▶ **Example 9** (Continued)**.** The reader can verify from Figure 1 that tuples of the form $(1, 2, x)$, where x can be any value, are $\mathcal{H}_0$-forbidden for R. Furthermore, notice that the tuple $(1, 2, \perp)$ is also $\mathcal{H}_0$-forbidden (in our construction $(1, 2, \perp)$ being $\mathcal{H}_0$-forbidden implies that any tuple of the form $(1, 2, x)$ is $\mathcal{H}_0$-forbidden).

Next we formalize the intuition of the above example. We say that a tuple t_1 defined over Y *subsumes* another tuple t_2 defined over Y if for any $y_j \in Y$, either $t_1[y_j] = \perp$ or $t_1[y_j] = t_2[y_j]$. Observe that if t_1 subsumes t_2 and t_1 is $\mathcal{H}$-forbidden, t_2 must be $\mathcal{H}$-forbidden as well. A tuple is *minimally $\mathcal{H}$-forbidden* if it is $\mathcal{H}$-forbidden and is not subsumed by any other $\mathcal{H}$-forbidden tuple. In our example, $(1, 2, 1)$ is subsumed by $(1, 2, \perp)$, so it is not minimally $\mathcal{H}_0$-forbidden, but the tuple $(1, 2, \perp)$ is. Lemma 10 stated below will be used to prove Theorem 7:

▶ **Lemma 10.** *Let $\mathcal{H} = (X, Y, E)$ be a bipartite graph, and R be a relation defined on X. Then, the set of all minimally $\mathcal{H}$-forbidden tuples of R has size at most $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$ and it can be computed in time $O(\phi(\mathcal{H})|R|)$.*

To prove the above lemma, we present an algorithm that encodes all the minimally $\mathcal{H}$-forbidden tuples of R in a rooted tree $T_{\mathcal{H}}(R)$. The tree has labels for both the nodes and the edges. More precisely, the label $L(v)$ of some node v is either a tuple in R or a special symbol $\perp^*$ (only the leaves can have label $\perp^*$), while the label of an edge of the tree is a pair of the form (y_j, a), where $y_j \in Y$ and $a \in \texttt{Dom}$. The labels of the edges are used to construct a set of $\mathcal{H}$-forbidden tuples that includes the set of all minimally $\mathcal{H}$-forbidden tuples as follows:

For each leaf node v with label $L(v) = \perp^*$, let $(y_{j_1}, a_{j_1}), \ldots, (y_{j_m}, a_{j_m})$ be the edge labels in the order they appear from the root to the leaf. Then, the tuple $\texttt{tup}(v)$ defined on Y as follows is an $\mathcal{H}$-forbidden tuple (but not necessarily minimally $\mathcal{H}$-forbidden):

$$\texttt{tup}(v)[y_j] = \begin{cases} a_j & \text{if } j \in \{j_1, \ldots, j_m\} \\ \perp & \text{otherwise} \end{cases}$$

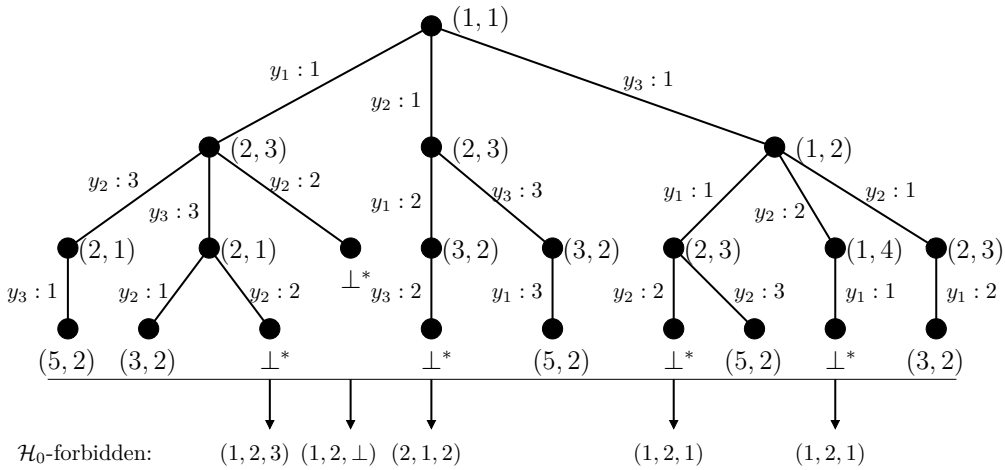


Figure 2 The tree $T_{\mathcal{H}}(R)$ of the running example. The diagram also presents how the $\mathcal{H}_0$-forbidden tuples are encoded by the tree.

Construction of $T_{\mathcal{H}}(R)$. We construct $T_{\mathcal{H}}(R)$ inductively by scanning through the tuples of R in an arbitrary order. As we read the next tuple t from R, we need to ensure that the $\mathcal{H}$-forbidden tuples that have been so far encoded by the tree are not $\mathcal{H}$-accepted by t: we achieve this by expanding some of the leaves and adding new edges and nodes to the tree. Therefore, after the algorithm has consumed a subset $R'' \subseteq R$, the partially constructed tree will be $T_{\mathcal{H}}(R'')$.

For the base of the induction, where $R'' = \emptyset$, we define $T_{\mathcal{H}}(\emptyset)$ as a tree that contains a single node (the root r) with label $L(r) = \perp^*$.

For the inductive step, let $T_{\mathcal{H}}(R'')$ be the current tree and let $t \in R$ be the next scanned tuple. The algorithm processes (in arbitrary order) all the leaf nodes v of the tree with $L(v) = \perp^*$. Let $(y_{j_1}, a_{j_1}), \ldots, (y_{j_p}, a_{j_p})$ be the edge labels in the order they appear on the path from root r to v. We distinguish two cases (for tuple t and a fixed leaf node v):

1. There exists $j \in \{j_1, \ldots, j_p\}$ and edge $(x_i, y_j) \in E$ such that $t[x_i] = a_j$. In this case, $\texttt{tup}(v)$ will be $\mathcal{H}$-forbidden in $R'' \cup \{t\}$; therefore, nothing needs to be done for this v.
2. Otherwise (*i.e.*, there is no such j), $\texttt{tup}(v)$ is not a $\mathcal{H}$-forbidden tuple for $R'' \cup \{t\}$. We set $L(v) = t$ (therefore, we never reassign the label of a node that has already been assigned to some tuple in R). There are two cases:
   a. If $p = \ell$, we cannot expand further from v (and will not expand in the future because now $L(v) \neq \perp^*$), since all y_j-s have been already set.
   b. If $p < \ell$, we expand the tree at node v. For every edge $(x_i, y_j) \in E$ such that $j \notin \{j_1, \ldots, j_p\}$, we add a fresh node $v^{i,j}$ with $L(v^{i,j}) = \perp^*$ and an edge $(v, v^{i,j})$ with label $(y_j, t[x_i])$. Notice that the tuples $\texttt{tup}(v^{i,j})$ will be now $\mathcal{H}$-forbidden in $R'' \cup \{t\}$.

The algorithm stops when either (a) all the tuples from R are scanned or (b) there exists no leaf node with label $\perp^*$.

▶ **Example 11** (Continued). We now illustrate the steps of the algorithm through the running example. After reading the first tuple, $t_1 = (1, 1)$, the algorithm expands the root node r to three children (for y_1, y_2, y_3), labels $L(r) = (1, 1)$ and labels the new edges as $(y_1, 1), (y_2, 1), (y_3, 1)$ and the new three leaves as $\perp^*$.

Suppose the second tuple $t_2 = (1, 2)$ is read next. First consider the leaf node with label $\perp^*$ that is reached from the root through the edge $(y_1, 1)$. At this point, the node

represents the tuple $(1, \perp, \perp)$. Observe that are in case (1) of the algorithm, and so the node is not expanded ($t_2[x_1] = 1$ and $m = 1 < 3 = \ell$). Consider now the third leaf node with label $\perp^*$, reached through the edge $(y_3, 1)$. We are now in case (2), and we have to expand the node. The available edges (since we have already assigned a value to y_3) are $(x_1, y_1), (x_1, y_2), (x_2, y_2)$. Hence, the node is labeled $(1, 2)$, and expands into three children, one for each of the above edges. These edges are labeled by $(y_1, 1), (y_2, 1), (y_2, 2)$ respectively; then the algorithm continues and at the end the tree in Figure 2 is obtained.

The $\mathcal{H}$-forbidden tuples encoded by the tree are not necessarily minimally $\mathcal{H}$-forbidden. However, for every minimally $\mathcal{H}$-forbidden tuple there exists a node in the tree that encodes it. In the running example, we find only two minimally $\mathcal{H}_0$-forbidden tuples for R: $(1, 2, \perp)$ and $(2, 1, 2)$. Furthermore, the constructed tree is not unique for R and depends on the order in which the tuples in R are scanned. The following lemma sums up the properties of the tree construction, and directly implies Lemma 10; the proof is deferred to the full version of the paper [15].

▶ **Lemma 12.** *$T_{\mathcal{H}}(R)$ satisfies the following properties:*

1. *The number of leaves is at most $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$.*
2. *Every leaf of $T_{\mathcal{H}}(R)$ with label $\perp^*$ encodes a $\mathcal{H}$-forbidden tuple.*
3. *Every minimally $\mathcal{H}$-forbidden tuple is encoded by some leaf of the tree with label $\perp^*$.*

For our running example, $\phi(\mathcal{H}_0) = 3! \cdot (1 \cdot 2 \cdot 1) = 12$, whereas the tree $T_{\mathcal{H}_0}(R)$ has only 10 leaves. We should note here that the bound $\phi(\mathcal{H})$ is tight, *i.e.* there exists an instance for which the number of minimally $\mathcal{H}$-forbidden tuples is exactly $\phi(\mathcal{H})$. [2]

We now discuss how we can use the tree $T_{\mathcal{H}}(R)$ to find a small $\mathcal{H}$-equivalent relation to R. It turns out that the connection is immediate: it suffices to collect the labels of all the nodes (not only leaves) of the tree $T_{\mathcal{H}}(R)$ that are not $\perp^*$. More formally:

$$\mathcal{E}_{\mathcal{H}}(R) = \{L(v) \mid v \in T_{\mathcal{H}}(R), L(v) \neq \perp^*\} \tag{1}$$

We can now show the following result, which completes the proof of Theorem 7:

▶ **Lemma 13.** *The set $\mathcal{E}_{\mathcal{H}}(R)$ is $\mathcal{H}$-equivalent to R and has size $|\mathcal{E}_{\mathcal{H}}(R)| \leq e \cdot \phi(\mathcal{H})$.*

▶ **Example 14** (Continued)**.** For our running example, the small $\mathcal{H}_0$-equivalent relation will be: $\mathcal{E}_{\mathcal{H}_0}(R) = \{(1,1), (1,2), (1,4), (2,1), (2,3), (3,2), (5,2)\}$. In other words, the tuples $(1,8), (2,2), (2,4), (10,2)$ are redundant and can be removed without affecting the answer to the query $(q_2, \mathcal{I})$.

Although the set of minimally $\mathcal{H}$-forbidden tuples is the same irrespective of the order by which the algorithm scans the tuples, the relation $\mathcal{E}_{\mathcal{H}}(R)$ depends on this order. It is an open problem to find the smallest possible $\mathcal{H}$-equivalent relation for R.

4 Query Plans for Inequalities

In this section, we use the techniques presented in the previous section as building blocks and prove Theorem 2. A *Select-Project-Join (SPJ) query plan* refers to a relational algebra expression that uses only selection (σ), projection (Π), and join ($\bowtie$) operators. Let $\mathcal{P}_q$ be

[2] For example, for $\mathcal{H}_0$ consider the instance $\{(1,2), (3,4), (5,6)\}$. The reader can check that the resulting tree has 12 leaves with label $\perp^*$, and that every leaf leads to a different minimally $\mathcal{H}$-forbidden tuple.

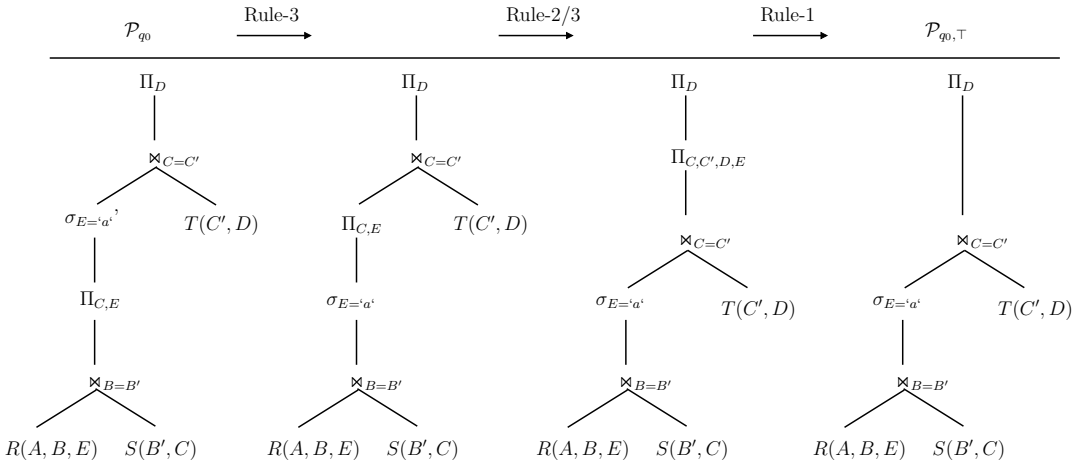


Figure 3 The relational plan $\mathcal{P}_{q_0}$ for Example 15, and the transformation to the plan $\mathcal{P}_{q_0,\top}$.

any SPJ query plan that computes a CQ q (without inequalities) on a database instance D in time $T(|q|, |D|)$. We will show how to transform $\mathcal{P}_q$ into a plan $\mathcal{P}_{q,\mathcal{I}}$ that computes $(q, \mathcal{I})$ in time $g(q, \mathcal{I}) \cdot \max(T(|q|, |D|))$. Without loss of generality, we assume that all the relation names and attributes in the base and derived relations (at intermediate steps in the plan) are distinct. Our running example for this section is given below:

▶ **Example 15.** Consider the query $(q_0, \mathcal{I})$, and the query plan $\mathcal{P}_{q_0}$ that computes q_0:

$$q_0(w) = R(x, y, `a`), S(y, z), T(z, w), \quad \mathcal{I} = \{x \neq z, y \neq w, x \neq w\}$$
$$\mathcal{P}_{q_0} = \Pi_D(\sigma_{E=`a`}(\Pi_{C,E}(R(A, B, E) \bowtie_{B=B'} S(B', C))) \bowtie_{C=C'} T(C', D))$$

The query plan $\mathcal{P}_{q_0}$ is depicted in Figure 3.

Clearly, this plan by itself does not work for $(q_0, \mathcal{I})$ as it is losing information that is essential to evaluate the inequlities, *e.g.*, $B(= B')$ is being projected out and it is used later in the inequality $x \neq w$ with the attribute C of T. To overcome this problem while keeping the same structure of the plan, we define a new projection operator that allows us to perform valid algebraic transformations, even in the presence of inequalities. Let $\text{att}(R)$ be the set of attributes that appear in a base or derived relation R; a query plan or sub-plan $\mathcal{P}$ is a derived relation with attributes $\text{att}(\mathcal{P})$. If $X \subseteq \text{att}(R)$, let $\bar{X}^R = \text{att}(R) \setminus X$.

▶ **Definition 16** ($\mathcal{H}$-Projection)**.** Let R be a base or a derived relation in $\mathcal{P}$. Let $X \subseteq \text{att}(R)$ and $\mathcal{H} = (\bar{X}^R, \text{att}(\mathcal{P}) \setminus \text{att}(R), E)$ be a bipartite graph. Then, the $\mathcal{H}$-projection of R on X, denoted $\Pi^{\mathcal{H}}_X(R)$, is defined as

$$\Pi^{\mathcal{H}}_X(R) = \bigcup_{\alpha \in \Pi_X(R)} \mathcal{E}_{\mathcal{H}}(\sigma_{X=\alpha}(R)) \tag{2}$$

where $\mathcal{E}_{\mathcal{H}}$ denotes an $\mathcal{H}$-equivalent subrelation as defined and constructed in equation (1).

Intuitively, $\mathcal{H}$ contains the inequalities between the attributes in $\bar{X}^R$ (that are being projected out) and the attributes of the rest of the query plan. The operator $\Pi^{\mathcal{H}}_X$ first groups the tuples from R according to the values of the X-attributes, but then instead of projecting out the values of the attributes in $\bar{X}^R$ for each such group, it computes a small $\mathcal{H}$-equivalent subrelation according to the graph $\mathcal{H}$.

▶ Observation 17. The $\mathcal{H}$-projection of a relation R on X satisfies the following properties:

1. $\Pi_X(R) = \Pi_X(\Pi_X^{\mathcal{H}}(R))$
2. $|\Pi_X^{\mathcal{H}}(R)| \leq e \cdot \phi(\mathcal{H}) \cdot |\Pi_X(R)|$ (ref. Lemma 7)

First step. To construct the plan $\mathcal{P}_{q,\mathcal{I}}$ from $\mathcal{P}_q$, we first create an equivalent query plan $\mathcal{P}_{q,\top}$ by pulling all the projections in $\mathcal{P}_q$ to the top of the plan. The equivalence of $\mathcal{P}_q$ and $\mathcal{P}_{q,\top}$ is maintained by the following standard algebraic rules regarding projections:

(Rule-1) Absorption: If $X \subseteq Y$, then $\Pi_X(R) = \Pi_X(\Pi_Y(R))$.
(Rule-2) Distribution: If $X_1 \subseteq \text{att}(R_1)$ and $X_2 = \text{att}(R_2)$, then $\Pi_{X_1 \cup X_2}(R_1 \times R_2) = \Pi_{X_1}(R_1) \times R_2$.
(Rule-3) Commutativity with Selection: If the selection condition θ is over a subset of X, then $\sigma_\theta(\Pi_X(R)) = \Pi_X(\sigma_\theta(R))$.

Figure 3 depicts how each rule is applied in our running example to transform the initial query plan $\mathcal{P}_{q_0}$ to $\mathcal{P}_{q_0,\top}$, where the only projection occurs in the top of the query plan. Observe that to distribute a projection over a join $R_1 \bowtie_{A_1=A_2} R_2$ (and not a cartesian product), we can write it as $\sigma_{A_1=A_2}(R_1 \times R_2)$, use both (Rule-2) and (Rule-3) to push the projection, and then write it back in the form as $R_1 \bowtie_{A_1=A_2} R_2$.

The plan $\mathcal{P}_{q,\top}$ will be of the form $\mathcal{P}_{q,\top} = \Pi_X(\mathcal{P}_0)$, where $\mathcal{P}_0$ is a query plan that contains only selections and joins. Notice that the plan $\Pi_X(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ correctly computes $(q, \mathcal{I})$, since it applies the inequalities before projecting out any attributes.[3] However, the running time is not comparable with that of $\mathcal{P}_q$ since the structures of the plans $\mathcal{P}_q$ and $\Pi_X(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ are very different. To achieve comparable running time, we modify $\Pi_X(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ by applying the corresponding rules of (Rule-1), (Rule-2), (Rule-3) for $\mathcal{H}$-projection in the reverse order.

Second step. To convert projections to $\mathcal{H}$-projections, first, we replace Π_X with $\Pi_X^{\mathcal{H}_0}$, where $\mathcal{H}_0 = (\text{att}(\mathcal{P}_0) \setminus X, \emptyset, \emptyset)$. Notice that $\Pi_X^{\mathcal{H}_0}$ is essentially like Π_X, but instead of removing the attributes that are not in X, the operator keeps an arbitrary witness. Thus, if we compute $\Pi_X^{\mathcal{H}_0}(\sigma_{\mathcal{I}}(\mathcal{P}_0))$, we not only get all tuples t in $(q, \mathcal{I})$, but for every such tuple we obtain a tuple t' from $(q^f, \mathcal{I})$ such that $t = t'[X]$. For our running example, $X = \{D\}$, and therefore, $\mathcal{H}_0 = (\{A, B, B', C, C', E\}, \emptyset, \emptyset)$ (see the rightmost plan in Figure 4).

Third step. We next present the rules for $\mathcal{H}$-projections to convert $\Pi_X^{\mathcal{H}_0}(\sigma_{\mathcal{I}}(\mathcal{P}_0))$ to the desired plan $\mathcal{P}_{q,\mathcal{I}}$. To show that the rules are algebraically correct, we need a weaker version of plan equivalence.

▶ **Definition 18** (Plan Equivalence)**.** Two plans $\mathcal{P}_1, \mathcal{P}_2$ are equivalent under $\Pi_X^{\mathcal{H}}$, denoted $\mathcal{P}_1 \equiv_X^{\mathcal{H}} \mathcal{P}_2$, if for every tuple α, $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\alpha}(\mathcal{P}_1))$ and $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\alpha}(\mathcal{P}_2))$ are $\mathcal{H}$-equivalent.

In other words, we do not need to have the same values of the attributes that are being projected out by Π_X in the small sub-relations $\mathcal{E}_{\mathcal{H}}$. We write $\mathcal{I}[X_1, X_2] \subseteq \mathcal{I}$ to denote the inequalities between attributes in subsets X_1 and X_2. For convenience, we also write $\mathcal{I}[X, X] = \mathcal{I}[X]$. We use $E[X_1, X_2]$ in a similar fashion, where E is the set of edges in a bipartite graph. Let $\mathbf{A} = \text{att}(\mathcal{P}_0)$. We apply the transformation rules for a sub-plan that is of the form $\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(S))$, where $\mathcal{I}$ is defined on $\text{att}(S)$ and $\mathcal{H} = (\bar{X}^S, \mathbf{A}, E)$. [4] The rules are:

[3] From here on we let $\mathcal{I}$ denote inequalities on attributes and not variables.
[4] For the sake of simplicity, we do not write the bipartite graph as $\mathcal{H} = (\bar{X}^S, \mathbf{A} \setminus \text{att}(S), E)$. However, the transformation rules ensure that the edges E in the bipartite graph are always between $\bar{X}^S$ and $\mathbf{A} \setminus \text{att}(S)$.

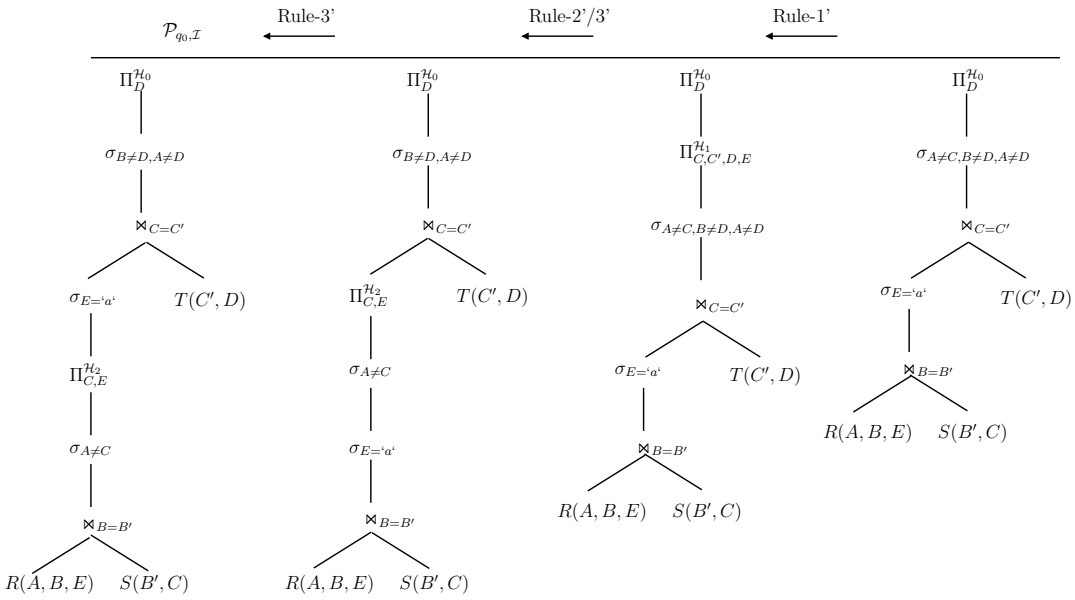


Figure 4 The reverse application of rules for Example 15. The bipartite graphs defined have edge sets $E(\mathcal{H}_0) = \emptyset$, $E(\mathcal{H}_1) = \emptyset$ and $E(\mathcal{H}_2) = \{(A, D), (B, D)\}$.

(Rule-1'). If $X \subseteq Y$ and $\mathcal{H}' = (\bar{Y}^S, \mathbf{A}, E[\bar{Y}^S, \mathbf{A}])$, then

$$\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(S)) \quad \equiv_X^{\mathcal{H}} \quad \Pi_X^{\mathcal{H}}(\Pi_Y^{\mathcal{H}'}(\sigma_{\mathcal{I}}(S)))$$

In the running example, we have $X = \{D\}$, $Y = \{C, C', D, E\}$, and $\text{att}(S) = \mathbf{A} = \{A, B, B', C, C', D, E\}$. The new bipartite graph for Rule-1' in Figure 4 (corresponding to Rule-1 in Figure 3) is $\mathcal{H}_1 = (\{A, B, B'\}, \mathbf{A}, \emptyset)$.

(Rule-2'). Let $S = R_1 \times R_2$, and $X = X_1 \cup Z_2$, where $X_1 \subseteq \text{att}(R_1) = Z_1$ and $Z_2 = \text{att}(R_2)$. If we define $\mathcal{H}' = (Z_1 \setminus X_1,\ \mathbf{A},\ E[Z_1 \setminus X_1, \mathbf{A}] \cup \mathcal{I}[Z_1 \setminus X_1, Z_2])$, then

$$\Pi_{X_1 \cup Z_2}^{\mathcal{H}}(\sigma_{\mathcal{I}}(R_1 \times R_2)) \quad \equiv_X^{\mathcal{H}} \quad \sigma_{\mathcal{I} \setminus \mathcal{I}[Z_1]}(\Pi_{X_1}^{\mathcal{H}'}(\sigma_{\mathcal{I}[Z_1]}(R_1)) \times R_2)$$

This rule adds new edges to the bipartite graph (which is initially empty) from the set of inequalities $\mathcal{I}$. In the running example, we have $X_1 = \{C, E\} \subseteq \{A, B, B', C, E\} = Z_1$ and $Z_2 = \{C', D\}$. Since $E(\mathcal{H}_1) = \emptyset$, to construct the edge set of the new bipartite graph $\mathcal{H}_2$, we need to find the inequalities that have one attribute in $Z_1 \setminus X_1 = \{A, B, B'\}$ and the other in $Z_2 = \{C', D\}$: these are $A \neq D$ and $B \neq D$. Hence, $\mathcal{H}_2 = (\{A, B, B'\}, \mathbf{A}, \{(A, D), (B, D)\})$, and the application of the rule is depicted in Figure 4.

(Rule-3'). If θ is defined over a subset of X, and $S = \sigma_\theta(R)$:

$$\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(\sigma_\theta(R))) \quad \equiv_X^{\mathcal{H}} \quad \sigma_\theta(\Pi_X^{\mathcal{H}}(\sigma_{\mathcal{I}}(R)))$$

In the running example, we move the selection operator $\sigma_{E='a'}$ before the projection operator $\Pi_{C,E}^{\mathcal{H}_2}$ as the last step of the transformation.

The proof of correctness of these transformations (*i.e.*, (Rule-1'), (Rule-2'), (Rule-3') preserve the equivalence of the plans under $\Pi_X^{\mathcal{H}}$) is deferred to the full version of the paper [15]. After applying the above transformations in the reverse order, the following lemma holds:

▶ **Lemma 19.** *Let $\mathcal{P}_q$ be an SPJ plan for q. For a set of inequalities $\mathcal{I}$, the transformed plan $\mathcal{P}_{q,\mathcal{I}}$ has the following properties:*

1. *If $\mathcal{P}_{q,\top} = \Pi_X(\mathcal{P}_0)$, the plan $\Pi_X(\mathcal{P}_{q,\mathcal{I}})$ computes $(q,\mathcal{I})$ (after projecting out the attributes that served as witness from $\mathcal{P}_{q,\mathcal{I}}$).*
2. *For every Π_X operator in $\mathcal{P}_q$, there exists a corresponding $\Pi_X^{\mathcal{H}}$ operator in $\mathcal{P}_{q,\mathcal{I}}$ for some appropriately constructed $\mathcal{H}$.*
3. *Every intermediate relation R in $\mathcal{P}_{q,\mathcal{I}}$ has size at most $e \cdot \max_{\mathcal{H}}\{\phi(\mathcal{H})\} \cdot |R'|$, where R' is the corresponding intermediate relation in $\mathcal{P}_q$.*
4. *If $T(|q|,|D|)$ is the time to evaluate $\mathcal{P}_q$, the the time to evaluate $\mathcal{P}_{q,\mathcal{I}}$ increases by a factor of at most $(e \cdot \max_{\mathcal{H}}\{\phi(\mathcal{H})\})^2$.*

Theorem 2 directly follows from the above lemma. To prove the bound on the running time, we use the fact that each operator (selection, projection or join) can be implemented in at most quadratic time in the size of the input (*i.e.*, $T(MN) \leq cM^2T(N)$). Additionally, notice that, if k is the vertex size of the inequality graph, then $\max_{\mathcal{H}}\{\phi(\mathcal{H})\} \leq k!k^k$. Hence, the running time can increase at most by a factor of $2^{O(k \log k)}$ when inequalities are added to the query. In our running example, $\phi(\mathcal{H}_0) = 1$, $\phi(\mathcal{H}_0) = 1$ and $\phi(\mathcal{H}_2) = 2$, hence the resulting intermediate relations in will be at most $2e$ times larger than the ones in $\mathcal{P}_{q_0}$.

The following query with inequalities is an example where our algorithm gives much better running time than the color-coding-based or treewidth-based techniques described in the subsequent sections.

▶ **Example 20.** Consider $P^k() = R_1(x_1,x_2), R_2(x_2,x_3), \cdots, R_k(x_k,x_{k+1})$ with inequalities $\mathcal{I} = \{x_i \neq x_{i+2} \mid i \in [k-1]\}$. Let $\mathcal{P}$ be the SPJ plan that computes this acyclic query in time $O(k|D|)$ by performing joins from left to right and projecting out the attributes as soon as they join. Then, the plan $\mathcal{P}_{\mathcal{I}}$ that is constructed has constant $\max_{\mathcal{H}}\{\phi(\mathcal{H})\}$; thus, $(P^k,\mathcal{I})$ can be evaluated in time $O(k|D|)$ as well.

▶ Remark. In this section we compared the running time of queries with inequalities with SPJ plans that compute the query without the inequalities. However, optimal algorithms that compute CQs may not use SPJ plans, as the recent worst-case optimal algorithms in [17, 22] show. These algorithms apply to conjunctive queries without projections, where any inequality can be applied at the end without affecting the asymptotic running time. However, there are cases where nonstandard algorithms for Boolean CQs run faster than SPJ algorithms, *e.g.* $q() = R(x_1,x_2), R(x_2,x_3), \ldots, R(x_{2k},x_1)$, can be computed in time $O(N^{2-1/k})$, where $N = |R|$. We show in the full version [15] that our techniques can be applied in this case as well. However, it is an open whether we can use them for any black-box algorithm.

5 Color-coding Technique and Generalization of Theorem 1

In this section, we will review the color-coding technique from [4] and use it to generalize Theorem 1 for arbitrary CQs with inequalities (*i.e.*, not necessarily acyclic queries)[5].

▶ **Theorem 21.** *Let q be a CQ that can be evaluated in time $T(|q|,|D|)$. Then, $(q,\mathcal{I})$ can be computed in time $2^{O(k \log k)} \cdot log(|D|) \cdot T(|q|,|D|)$ where k is the number of variables in $\mathcal{I}$.*

[5] The $\log^2(|D|)$ factor in Theorem 1 is reduced to $\log(|D|)$ in Theorem 21, but this is because one log factor was due to sorting the relations in the acyclic query, and now this hidden in the term $T(|q|,|D|)$.

First, we state the original randomized color-coding technique to describe the intuition: randomly color each value of the active domain by using a hash function h, use these colors to check the inequality constraints, and use the actual values to check the equality constraints.

For a CQ q, let q^f denote the *full query* (without inequalities), where every variable in the body appears in the head of the query q. For a variable x_i and a tuple t, $t[x_i]$ (or simply $t[i]$ where it is clear from the context) denotes the value of the attribute of t that corresponds to variable x_i. Let $t \in q^f(D)$. We say that t *satisfies the inequalities* $\mathcal{I}$, denoted by $t \models \mathcal{I}$, if for each $x_i \neq x_j$ in $\mathcal{I}$, $t[x_i] \neq t[x_j]$. We say that t *satisfies the inequalities* $\mathcal{I}$ *with respect to the hash function* h, denoted by $t \models_h \mathcal{I}$, if for each such inequaity $h(t[x_i]) \neq h(t[x_j])$.

Recall that k is the number of variables that appear in $\mathcal{I}$. Let h be a perfectly random hash function $h : \mathtt{Dom} \to [p]$ (where $p \geq k$). For any $t \in q^f(D)$ if t satisfies $\mathcal{I}$, then with high probability it also satisfies $\mathcal{I}$ with respect to h, *i.e.*, $\mathbf{Pr}_h[t \models_h \mathcal{I} \mid t \models \mathcal{I}] \geq \frac{p(p-1)\cdots(p-k+1)}{p^k}$ $\geq e^{-2\sum_{i=1}^{k-1}(i/p)} \geq e^{-k}$, where we used the fact that $1 - x \geq e^{-2x}$ for $x \leq \frac{1}{2}$. Therefore, by repeating the experiment $2^{O(k)}$ times we can evaluate a Boolean query with constant probability.

This process can be derandomized leading to a deterministic algorithm (for evaluating any CQ, not necessarily Boolean) by selecting h from a family $\mathcal{F}$ of k-perfect hash functions. A k-perfect family guarantees that for every tuple of arity at most k (with values from the domain $\mathtt{Dom}$), there will be some $h \in \mathcal{F}$ such that for all $i, j \in [k]$, if $t[i] \neq t[j]$, then $h(t[i]) \neq h(t[j])$ (and thus if $t \models \mathcal{I}$, then $t \models_h \mathcal{I}$) It is known (see [4]) that we can construct a k-perfect family of size $|\mathcal{F}| = 2^{O(k)} \log(|\mathtt{Dom}|) = 2^{O(k)} \log |D|$.[6]

A coloring $\mathbf{c}$ of the vertices of the inequality graph $G^{\mathcal{I}}$ with k colors is called a *valid k-coloring*, if for each $x_i \neq x_j$ we have that $c_i \neq c_j$ where c_i denotes the color of variable x_i under $\mathbf{c}$. Let $\mathcal{C}(G^{\mathcal{I}})$ denote all the valid colorings of $G^{\mathcal{I}}$. For each such coloring $\mathbf{c}$ and any given hash function $h : \mathtt{Dom} \to [k]$, we can define a subinstance $D[\mathbf{c}, h] \subseteq D$ such that for each relation R, $R^{D[\mathbf{c},h]} = \{t \in R^D \mid \forall x_i \in vars(R), h(t[x_i]) = c_i\}$. In other words, the subinstance $D[\mathbf{c}, h]$ picks only the tuples that under the hash function h agree with the coloring $\mathbf{c}$ of the inequality graph. Then the algorithm can be stated as follows:

> **Deterministic Algorithm:** *For every hash function $h : \mathtt{Dom} \to [k]$ in a k-perfect hash family $\mathcal{F}$, for every valid k-coloring $\mathbf{c} \in \mathcal{C}(G^{\mathcal{I}})$ of the variables, evaluate the query q on the sub-instance $D[\mathbf{c}, h]$. Output $\bigcup_{h \in \mathcal{F}} \bigcup_{\mathbf{c} \in \mathcal{C}(G^{\mathcal{I}})} q(D[\mathbf{c}, h])$.*

The correctness argument for the above algorithm is presented in [15]. The running time of the algorithm is $O(|\mathcal{F}| \cdot |\mathcal{C}(G^{\mathcal{I}})| \cdot T(q, |D|))$. Since $|\mathcal{F}| \leq 2^{O(p)} \log |D|$ and $|\mathcal{C}(G^{\mathcal{I}})| \leq k^k$, Theorem 21 follows.

Comparison of Theorem 2 with Theorem 21. The factors dependent on the query in these two theorems ($g(q, \mathcal{I})$ in Theorem 2 and $f(k)$ in Theorem 21) are both bounded by $2^{O(k \log k)}$. However, our technique outperforms the color-coding technique in several respects. First, the randomized color-coding technique is simple and elegant, but is unsuitable to implement in a database system that typically aims to find deterministic answers. On the other hand, apart from the additional $\log(|D|)$ factor, the derandomized color-coding technique demands the construction of a new k-perfect hash family for every database instance and query, and therefore may not be efficient for practical purposes. Our algorithm requires no preprocessing and can be applied in a database system by maintaining the same query plan and using

[6] Assuming $\mathtt{Dom}$ includes only the attributes that appear as variables in the query q, $|\mathtt{Dom}| \leq |D||q|$.

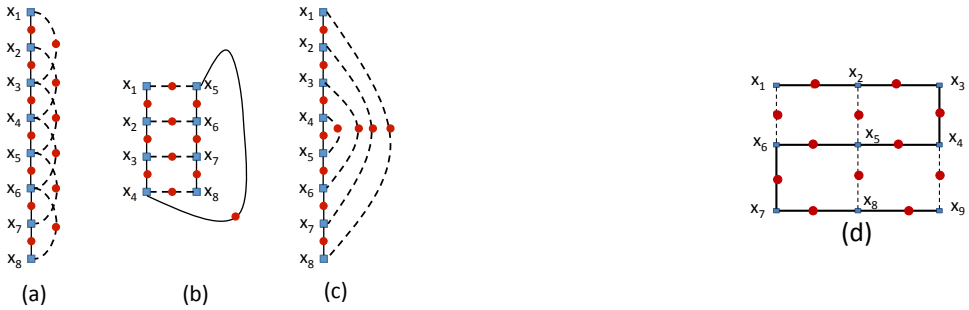


Figure 5 Augmented graphs for Example 22 ($k = 7$) and Example 23 ($k = 8$). The solid and dotted edges come from the query and inequalities respectively; the blue squares denote variables, and red circles denote (unnamed) relational atoms: (a) $(P^7, \mathcal{I}_1)$, (b) $(P^7, \mathcal{I}_2)$, (c) $(P^7, \mathcal{I}_3)$, (d) $(P^7, \mathcal{I}_4)$.

a more sophisticated projection operation. More importantly, the color coding technique is *oblivious* of the combined structure of the query and the inequalities. As an example, consider the path query P^k, together with the inequalities $\mathcal{I}_1 = \{x_i \neq x_{i+2} : i \in [k-1]\}$. The color-coding-based algorithm has a running time of $2^{O(k \log k)}|D| \log |D|$. However, as discussed in Section 4, we can compute this query in time $O(k|D|)$, thus the exponential dependence on k is eliminated.

6 CQs and Inequalities with Polynomial Combined Complexity

In this section, we investigate classes of queries and inequalities that entail a poly-time combined complexity for $(q, \mathcal{I})$ in terms of the treewidths of query graph G^q, inequality graph $G^{\mathcal{I}}$, and augmented graph $G^{q,\mathcal{I}}$. If the augmented graph $G^{q,\mathcal{I}}$ has bounded treewidth, then $(q, \mathcal{I})$ can be answered in poly-time combined complexity [23, 6]. We give examples of such q and $\mathcal{I}$ below:

▶ **Example 22.** Consider the path query: $P^k(\ ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$, which is acyclic, and consider the following inequality patterns (see Figure 5): (i) $(P^k, \mathcal{I}_1)$ where $\mathcal{I}_1 = \{x_i \neq x_{i+2} : i \in [k-1]\}$ has treewidth 2. (ii) $(P^k, \mathcal{I}_2)$ where $\mathcal{I}_2 = \{x_i \neq x_{i+\frac{k}{2}} : i \in [\frac{k+1}{2}]\}$ has treewidth 3 (k is odd). (iii) $(P^k, \mathcal{I}_3)$ where $\mathcal{I}_3 = \{x_i \neq x_{k-i+1} : i \in [\frac{k+1}{2}]\}$ has treewidth 2 (k is odd).

However, for certain inputs our algorithm in Section 4 can outperform the treewidth-based techniques since it considers the inequality structure more carefully. For instance, even though the augmented graph of $(P^k, \mathcal{I}_1)$ has treewidth 2 (see Figure 5 (a)), the techniques of [23] will give an algorithm with running time $O(poly(k)|D|^2)$, whereas the algorithm in Section 4 gives a running time of $O(k|D|)$.

Indeed, the treewidth of $G^{q,\mathcal{I}}$ is at least as large as the treewidth of G^q and $G^{\mathcal{I}}$. As mentioned earlier, when $G^{\mathcal{I}}$ is the complete graph on $k+1$ variables (with treewidth $= k+1$), answering $(P^k, \mathcal{I})$ is as hard as finding if a graph on $k+1$ vertices has a Hamiltonian path, and therefore is NP-hard in k. Interestingly, even when both G^q and $G^{\mathcal{I}}$ have bounded treewidths, $G^{q,\mathcal{I}}$ may have unbounded treewidth as illustrated by the following example:

▶ **Example 23.** Consider $(P^k, \mathcal{I}_4)$ (see Figure 5(d)), where $k+1 = p^2$ for some p. Algebraically, we can write $\mathcal{I}_4$ as: $\mathcal{I}_4 = \{x_i \neq x_{\lfloor i/p \rfloor + 1 + 2p - (i \mod p)} \mid i = 1, \ldots, p(p-1)\}$. The edges for P^k are depicted in the figure as an alternating path on the grid with solid edges, whereas

the remaining edges are dotted and correspond to the inequalities. Here both G^{P^k} and $G^{\mathcal{I}_4}$ have treewidth 1, but $G^{P^k,\mathcal{I}_4}$ has treewidth $\Theta(\sqrt{k})$.

However, this does not show that evaluation of the query $(P^k, \mathcal{I}_4)$ is NP-hard in k, which we prove below by a reduction from the *list coloring problem*:

▶ **Definition 24** (List Coloring)**.** Given an undirected graph $G = (V, E)$, and a list of admissible colors $L(v)$ for each vertex $v \in V$, list coloring asks whether there exists a coloring $c(v) \in L(v)$ for each vertex v such that the adjacent vertices in G have different colors.

The list coloring problem generalizes the coloring problem, and therefore is NP-hard. List coloring is NP-hard even on grid graphs with 4 colors and where $2 \leq |L(v)| \leq 3$ for each vertex v [7]; we show NP-hardness for $(P^k, \mathcal{I}_4)$ by a reduction from list coloring on grids.

▶ Proposition 25. The combined complexity of evaluating $(P^k, \mathcal{I}_4)$ is NP-hard, where both the query P^k and the inequality graph G are acyclic (have treewidth 1).

In fact, the above proposition can be generalized as follows: *if the graph $G^{q,\mathcal{I}}$ is NP-hard for list coloring for a query q where each relation has arity 2, then evaluation of the query $(q, \mathcal{I})$ is also NP-hard in the size of the query.*

On the contrary, $(q, \mathcal{I})$ may not be hard in terms of combined complexity if the treewidth of $G^{q,\mathcal{I}}$ is unbounded, which we also show with the help of the list coloring problem. Consider the queries $F^k(\ ) = R_1(x_1), R_2(x_2), \ldots, R_k(x_k)$. Given inequalities $\mathcal{I}$, the evaluation of $(F^k, \mathcal{I})$ is *equivalent* to the list coloring problem on the graph $G^{\mathcal{I}}$ when the available colors for each vertex x_i are the tuples in $R_i(x_i)$. Since list coloring is NP-hard:

▶ Proposition 26. The evaluation of $(F^k, \mathcal{I})$ is NP-hard in k for arbitrary inequalities $\mathcal{I}$.

Therefore, answering $(F^k, \mathcal{I})$ becomes NP-hard in k even for this simple class of queries if we allow arbitrary set of inequalities $\mathcal{I}$ (this also follows from Theorem 29). However, list coloring can be solved in polynomial time for certain graphs $G^{\mathcal{I}}$: (i) **Trees** (the problem can be solved in time $O(|V|)$ independent of the available colors[13]), and in general graphs of constant treewidth. (ii) **Complete graphs** (by a reduction to *bipartite matching*).[7] In general, if the connected components of G are either complete graphs or have constant treewidth, list coloring can be solved in polynomial time. Therefore, on such graphs as $G^{\mathcal{I}}$, the query $(F^k, \mathcal{I})$ can be computed in poly-time in k and $|D|$. Here we point out that none of the other algorithms given in this paper can give a poly-time algorithm in $k, |D|$ for $(F^k, \mathcal{I})$ when $G^{\mathcal{I}}$ is the complete graph (and therefore has treewidth k). The following proposition generalizes this property:

▶ Proposition 27. Let q be a Boolean CQ, where each relational atom has arity at most 2. If q has a *vertex cover* (a set of variables that can cover all relations in q) of constant size and the list coloring problem on $G^{\mathcal{I}}$ can be solved in poly-time, then $(q, \mathcal{I})$ can be answered in poly-time combined complexity.

The proof is given in the full version of the paper. To see an example, consider the star query $Z^n(\ ) = R_1(y, x_1), \ldots, R_n(y, x_n)$ which has a vertex cover $\{y\}$ of size 1. We iterate over all possible values of y: for each such value $\alpha \in \texttt{Dom}$, the query $R_1(\alpha, x_1), \ldots, R_n(\alpha, x_n)$ is equivalent to F^n, and therefore $(Z^n, \mathcal{I})$ can be evaluated in poly-time in combined complexity when $G^{\mathcal{I}}$ is an easy instance of list coloring.

[7] We can construct a bipartite graph where all vertices v appear on one side, the colors appear on the other side, and there is an edge (v, c) if $c \in L(v)$. Then the list coloring problem on complete graph is solvable if and only if there is a perfect matching in the graph.

7 CQs with Polynomial Combined Complexity for All Inequalities

This section aims to find CQs q such that computing $(q, \mathcal{I})$ has poly-time combined complexity, no matter what the choice of $\mathcal{I}$ is. Here we present a sufficient condition for this, and a stronger necessary condition.

A *fractional edge cover* of a CQ q assigns a number v_R to each relation $R \in q$ such that for each variable x, $\sum_{R:x\in vars(R)} v_R \geq 1$. A *fractional vertex packing* (or, *independent set*) of q assigns a number u_x to each variable x, such that $\sum_{x\in vars(R)} u_x \leq 1$ for every relation $R \in q$. By duality, the minimum fractional edge cover is equal to the maximum fractional vertex packing. When each $v_R \in \{0, 1\}$ we get an *integer edge cover*, and when each $u_x \in \{0, 1\}$ we get an *integer vertex packing*.

▶ **Definition 28.** A family $\mathcal{Q}$ of Boolean CQs has *unbounded fractional (resp. integer) vertex packing* if there exists a function $T(n)$ such that for every integer $n > 0$ it can output in time $poly(n)$ a query $q \in \mathcal{Q}$ that has a fractional (resp. integer) vertex packing of size at least n (counting relational atoms as well as variables).

A family $\mathcal{Q}$ of Boolean CQs has *bounded fractional (resp. integer) vertex packing* if there exists a constant $b > 0$ such that for any $q \in \mathcal{Q}$, the size of any fractional (resp. integer) vertex packing is $\leq b$.

Path queries $P^k( ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_{k+1})$ and cycle queries $C^k( ) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_1)$ are examples of classes of unbounded vertex packing.

▶ **Theorem 29.**

1. *If a family of Boolean CQs $\mathcal{Q}$ has unbounded integer vertex packing, the combined complexity of $(q, \mathcal{I})$ for $q \in \mathcal{Q}$ is NP-hard.*
2. *If a family of CQs $\mathcal{Q}$ has bounded fractional vertex packing, then for each $q \in \mathcal{Q}$, $(q, \mathcal{I})$ can be evaluated in poly-time combined complexity for any $\mathcal{I}$.*

The NP-hardness in this theorem follows by a reduction from 3-Coloring, whereas the poly-time algorithm uses the bound given by Atserias-Grohe-Marx [12, 5] in terms of the size of minimum fractional edge cover of the query, and the duality between minimum fractional edge cover and maximum fractional vertex packing. The formal proof of the above theorem will appear in the full version of the paper.

In this paper, we illustrate the properties with examples. Consider the family $S^k( ) = R(x_1, \ldots, x_k)$ for $k \geq 1$: this has vertex packing of size $= 1$ and therefore can be answered trivially in poly-time in combined complexity for any inequality pattern $\mathcal{I}$. On the other hand, the class of path queries P^k mentioned earlier has unbounded vertex packing (has a vertex packing of size $\approx \frac{k}{2}$), and therefore for certain set of inequalities (*e.g.*, when $G^{\mathcal{I}}$ is a complete graph), the query evaluation of $(P^k, \mathcal{I})$ is NP-hard in k. Similarly, the class $F^k( ) = R_1(x_1), R_2(x_2), \ldots, R_k(x_k)$ mentioned earlier has unbounded vertex packing, and is NP-hard in k with certain inequality patterns (see Proposition 26).

Theorem 29 is not a dichotomy or a characterization of easy CQs w.r.t. inequalities, since there is a gap between the maximum fractional and integer vertex packing.[8]

[8] For example, for the complete graph on k vertices, the maximum integer vertex packing is of size 1 whereas the maximum fractional vertex packing is of size $\frac{k}{2}$.

8 Conclusion

We studied the complexity of CQs with inequalities and compared the complexity of query answering with and without the inequality constraints. Several questions remain open: Is there a property that gives a dichotomy of query evaluation with inequalities both for the class of CQs, and for the class of CQs along with the inequality graphs? What can be said about unions of conjunctive queries (UCQ) and recursive datalog programs? Can our techniques be used as a black-box to extend any algorithm for CQs, *i.e.*, not necessarily based on SPJ query plans, to evaluate CQs with inequalities?

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Foto Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. In *PODS*, pages 209–220, 2002.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

5 Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *FOCS*, pages 739–748, 2008.

6 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

7 Marc Demange and Dominique De Werra. On some coloring problems in grids. *Theor. Comput. Sci.*, 472:9–27, February 2013.

8 Arnaud Durand and Etienne Grandjean. The complexity of acyclic conjunctive queries revisited. *CoRR*, abs/cs/0605008, 2006.

9 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, November 2002.

10 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. In *PODS*, pages 21–32, 1999.

11 M.H. Graham. On the universal relation. *Technical Report, University of Toronto, Ontario, Canada*, 1979.

12 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

13 Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997.

14 Phokion G. Kolaitis, David L. Martin, and Madhukar N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *PODS*, pages 197–204, 1998.

15 Paraschos Koutris, Tova Milo, Sudeepa Roy, and Dan Suciu. Answering conjunctive queries with inequalities. *CoRR*, abs/1412.3869, 2014.

16 B. Monien. How to find long paths efficiently. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 239 – 254. North-Holland, 1985.

17 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48, 2012.

18 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. In *PODS*, pages 12–19, 1997.

19 Neil Robertson and P.D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

20 Riccardo Rosati. The limits of querying ontologies. In *ICDT*, pages 164–178, 2007.

21 Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.*, 54(1):113–135, February 1997.

22 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 96–106, 2014.

23 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94. IEEE Computer Society, 1981.

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

25 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discrete Math.*, 10(2):209–222, 1997.