Theory Comput Syst (2017) 61:2–30
DOI 10.1007/s00224-016-9684-2

Answering Conjunctive Queries with Inequalities

Paraschos Koutris[1] · Tova Milo[2] · Sudeepa Roy[1] · Dan Suciu[1]

Published online: 7 June 2016

Abstract In this paper, we study the complexity of answering conjunctive queries (CQ) with inequalities ($\neq$). In particular, we are interested in comparing the complexity of the query with and without inequalities. The main contribution of our work is a novel combinatorial technique that enables us to use 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. The key idea is to define a new projection operator, which 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 and Yannakakis (1997) and give an alternative algorithm based on the color-coding technique (2008) 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

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 by the Israel Ministry of Science.

✉ Paraschos Koutris
pkoutris@cs.washington.edu

[1] University of Washington, Seattle WA, USA

[2] Tel Aviv University, Tel Aviv, Israel

necessary properties and sufficient properties that allow a class of CQs to have polynomial-time combined complexity with respect to any inequality pattern. We also illustrate classes of queries where our query-plan-based technique outperforms the alternative approaches discussed in the paper.

Keywords Query evaluation · Conjunctive query · Inequality · Treewidth

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* (where both the query and the database are inputs), while the *data complexity* of CQs (where the query is fixed) is in AC_0 [1]. Yannakakis [24] showed that evaluation of acyclic CQs has polynomial-time combined complexity. This result was later generalized to CQs with bounded treewidth, bounded query-width, or bounded hypertreewidth: the combined complexity remains polynomial if the width of a tree or query decomposition of the query hypergraph is bounded [6, 9, 10, 16].

However, the complexity of query evaluation c;hanges drastically once we add inequalities in the body of the query. Consider the following Boolean acyclic family of CQs P^k:

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

This query can be computed in $O(k|D|)$ time on an instance D by evaluating the joins from left to right, since at any intermediate step that we join with relation R_j we only need to keep a projection on variable x_j. If we add the inequalities $x_i \neq x_j$ for every $i < j$ and evaluate it on an instance where each R_ℓ, $1 \leq \ell \leq k$, 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 [21] observed this fact and showed that the problem is *fixed-parameter tractable* for acyclic CQs:

Theorem 1 ([21]) *Let q be an acyclic CQ 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 [19] 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, 4, 26].

In the context of databases, Klug [15] studied the problems of query containment and equivalence for CQs with comparison operators between variables ($<, \leq$ etc.) under the assumption that the data domains are dense (e.g., rational numbers) and totally ordered, and showed that these problems are in Π_2^p. Papadimitriou and Yannakakis [21] showed that answering acyclic CQs with comparison operators 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 [18]; the effect of several syntactic properties of Q_1, Q_2 on the complexity of this problem has been studied by Kolaitis et al. [16]. Durand and Grandjean [8] improved Theorem 1 from [21] 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].

From a practical point of view, the query optimizers of state-of-the-art database systems do not use any specific techniques to optimize for inequality conditions in the WHERE clause. In most cases, query optimizers do not even use available indexes for applying the inequality conditions, since scanning the table is almost always a cheaper alternative. The most relevant work to our setting is evaluation of queries with *inequality joins* [14], where the join has conditions of the form $R.A \leq S.B$.

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.

Consider any *Select-Project-Join (SPJ) query plan* for a given CQ, which is simply a relational algebra expression that uses only selection (σ), projection (Π), and join ($\bowtie$) operators (see for example Fig. 3). The main result in this paper says that any such SPJ 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, we can construct a query plan $\mathcal{P}_{q,\mathcal{I}}$ that evaluates the query $(q, \mathcal{I})$ in time $g(q, \mathcal{I}) \cdot \max(T(|q|, |D|), |D|)$, where g is a function that is independent of the input database.*[1]

[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.

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. **(Sections 3, 4)** First, we give the proof of Theorem 2. 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 6**) 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 6 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 6, 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 1**). 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 3**).
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 7**).

In addition, in Section 2, we review some useful notions (e.g., tree decomposition, treewidth, acyclicity of a query) and define query, inequality, and augmented graphs that are used in the rest of the paper; we conclude in Section 8 with directions of future research.

Comparison with the conference version [17] This paper is an extended version of [17], which was published in the International Conference on Database Theory (ICDT), 2015. The following material has been added in this journal submission:

1. The proof of Lemma 2.
2. The proof of Lemma 3.
3. The proof of Lemma 4.
4. The algorithm for the cycle query C_{2k} with inequalities, presented at the end of Section 4.

5. The proof of Theorem 6.
6. The proof of Theorem 1.
7. The proof of Proposition 3.
8. The proof of Theorem 7.

2 Preliminaries

We are given a CQ q, a set of inequalities $\mathcal{I}$, and a database instance D. We will use $vars(q)$ to denote the variables in the body of query q and $\texttt{Dom}$ 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*. For a CQ q, let q^f denote the full query where every variable in the body of q appears in the head of the query q.

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 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 $\mathcal{I}$ ($k \leq |vars(q)| < |q|$).

Given q and $\mathcal{I}$, the goal is to evaluate the query q with inequality $\mathcal{I}$, called the *augmented query* $(q, \mathcal{I})$, on D. Intuitively, the query $(q, \mathcal{I})$ augments 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 $a \neq b$.

Query Graph, Inequality Graph, and Augmented Graph Given a CQ q and a set of inequalities $\mathcal{I}$, we define three undirected graphs:

1. **Query incidence graph** ($\mathbf{G^q}$): 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.
2. **Inequality graph** ($\mathbf{G^{\mathcal{I}}}$): The *inequality graph* contains the variables in $vars(q)$ as the set of vertices. $G^{\mathcal{I}}$ adds an edge between $x_i, x_j \in vars(q)$ if the inequality $x_i \neq x_j$ belongs to $\mathcal{I}$.
3. **Augmented graph** ($\mathbf{G^{q,\mathcal{I}}}$): The *augmented graph* $G^{q,\mathcal{I}}$ contains the variables in $vars(q)$ as the set of vertices too, and is the query incidence graph of the augmented query $(q, \mathcal{I})$.

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 1 (Treewidth) A *tree decomposition* [22] 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, 25] of a query repeatedly removes *ears* from the query hypergraph (hyperedges having at least one variable that does not belong to any other hyperedge and there exists some other hyper edge that contains all other variables) until no further ears exist.

Definition 2 A query is *acyclic* if its GYO-reduction is the empty hypergraph, otherwise it is cyclic.

As an 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: we can continue removing one of the two ears from the two ends (in the first step, R_1 or R_k) until the hypergraph becomes empty. On the other hand, $C^k(\,) = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_1)$ is a cyclic query (no ears in the query hypergraph).

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 [6, 9, 10, 16, 24].

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 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. Several of our ideas and techniques are related to the work of [8, 19].[2]

We now formalize the above intuition. Let $X = \{x_1, \cdots, x_m\}$ and $Y = \{y_1, \cdots, y_\ell\}$. 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 $t[x_i]$ denote the value of the attribute of t that corresponds to variable x_i.

Definition 3 ($\mathcal{H}$-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 $t_R[x_i] \neq t[y_j]$.

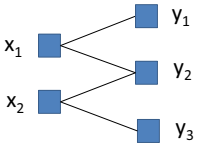
Notice that $(q_2, \mathcal{I})$ is true if and only if there exists a tuple $t_S \in S$ that is $\mathcal{H}$-accepted by R.

Example 1 (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 Fig. 1a and consider the instance for R as depicted in Fig. 1b. 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 4 ($\mathcal{H}$-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 every 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.

[2] Monien in [19] defines the notion of *q-representatives* for families of sets. Given a family of sets F, where each set has p elements, $\hat{F} \subseteq F$ is a q-representative if for every set T of size q, there exists some set $U \in F$ with $U \cap T = \emptyset$ if and only if there exists a set $\hat{U} \in \hat{F}$ such that $\hat{U} \cap T = \emptyset$. Observe that a q-representative is a special case of an $\mathcal{H}$-equivalent relation: indeed, we can model the family F as a relation R^F of arity p (where we do not care about the order of the attributes), and define $\mathcal{H}$ as the full bipartite graph with edge set $[p] \times [q]$. Then, if we write $\mathcal{E}_{\mathcal{H}}(R^F)$ back to a family of sets, it is a q-representative of F. Our techniques also generalize the notion of *minimum samples* presented in [8], which corresponds to $\mathcal{H}$-forbidden tuples of a relation in the case where $\mathcal{H} = (X, Y, E)$ has $|X| = |Y|$ and $E(\mathcal{H})$ forms a perfect matching between X and Y. Several of the definitions and algorithmic ideas were inspired by both [8, 19].



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

(a) The bipartite graph $\mathcal{H}_0$ (b) The instance of $R(x_1, x_2)$

Fig. 1 The running example (Example 1) for Section 3

$\mathcal{H}$-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 3 *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. *$|R'| \leq e \cdot \phi(\mathcal{H})$, where e is Euler's number.*
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 $\mathtt{Dom}$.

Definition 5 ($\mathcal{H}$-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 $\mathtt{Dom} \cup \{\perp\}$ is *$\mathcal{H}$-forbidden* for R if for every 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 2 (Continued) The reader can verify from Fig. 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 1 stated below will be used to prove Lemma 3:

Lemma 1 *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):

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

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 exactly $T_{\mathcal{H}}(R'')$.

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 3 (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). It labels $L(r) = (1, 1)$, 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 we 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 Fig. 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 1.

Lemma 2 *$T_{\mathcal{H}}(R)$ 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 $\perp^*$.*

Proof We start by showing item (1). The first observation is that the depth of the tree is at most ℓ. Indeed, consider any path from the root to a leaf, and let

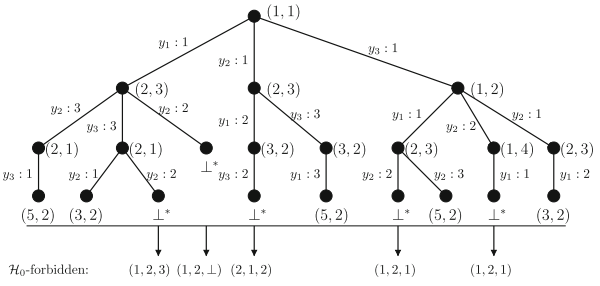


Fig. 2 The tree $T_{\mathcal{H}}(R)$ of the running example. The diagram also presents how the $\mathcal{H}_0$-forbidden tuples are encoded by the tree. We write $y_i : a$ to alternatively denote the edge label (y_i, a)

$(y_{j_1}, a_{j_1}), \ldots, (y_{j_m}, a_{j_m})$ be the labels of the edges. By the construction in step (2), all j_a are pairwise disjoint, and so we can have at most ℓ such labels in the path. Notice additionally that each such path visits a subset of the nodes in Y in some order, and maps each node it to one of its neighbors in X. This implies that the number of leaves in $T_{\mathcal{H}}(R)$ can be at most $\phi(\mathcal{H}) = \ell! \prod_{j \in [\ell]} d_{\mathcal{H}}(y_j)$.

Item (2) is straightforward and follows by the fact that only the expansion step (2) of the algorithm can assign the label $\perp^*$ to a node.

Finally, we prove item (3). Let t be a minimally $\mathcal{H}$-forbidden tuple. We will show that the algorithm will produce t at some leaf of the tree. Our argument will trace t along a path from the root of $T_{\mathcal{H}}(R)$ to the appropriate leaf.

Consider the tuples of R in the order visited by the algorithm: $t_1, t_2, \ldots$. We will show the following inductive statement: for each tuple t_a, there exists a leaf node v_a in the tree with label $\perp^*$ such that $\texttt{tup}(v_a)$ is $\mathcal{H}$-forbidden for $\{t_1, \ldots, t_a\}$ and subsumes t. This statement suffices to prove (3), since at the point where $a = |R| = m$ (i.e., all the tuples in R have been scanned), $\texttt{tup}(v_a)$ must equal t (otherwise t is not minimal), and also $L(v_a) = \perp^*$.

The statement vacuously holds before no tuples from R have been scanned for the root node that encodes $(\perp, \cdots, \perp)$, and forms the basis of the induction. Now, suppose that we are at some tuple t_a and node v_a where the inductive statement holds. Let t_{a+1} be the next tuple in the order. If the algorithm falls into case (1), then $v_{a+1} = v_a$ and $\texttt{tup}(v_{a+1}) = \texttt{tup}(v_a)$. Since $\texttt{tup}(v_a)$ is $\mathcal{H}$-forbidden for $t_1, \ldots, t_a$ and subsumes t, it will be $\mathcal{H}$-forbidden for $t_1, \ldots, t_{a+1}$ as well, and still subsume t. Further, the label of $v_{a+1} = v_a$ remains $\perp^*$.

Now suppose we fall into case (2) and t_{a+1} is read. Let $y_{j_1}, \ldots, y_{j_p}$ be the variables set so far in $\texttt{tup}(v_a)$ where v_a is labeled $\perp^*$. First note that we cannot fall into case (2a), i.e. $p < \ell$. Indeed, if $p = \ell$ and t_{a+1} satisfies all inequalities with $\texttt{tup}(v_a)$, then $\texttt{tup}(v_a)$ is not $\mathcal{H}$-forbidden. Since all the positions of $\texttt{tup}(v_a)$ have been set and $\texttt{tup}(v_a)$ subsumes t, it must hold that $\texttt{tup}(v_a) = t$. It follows that t is not $\mathcal{H}$-forbidden, which contradicts the fact that t remains $\mathcal{H}$-forbidden after all tuples in R are read.

Therefore, we are in case (2b), and for all $s \in [p]$, there exists some $x_i \in E(\mathcal{H})$, $t_{a+1}[x_i] \neq \texttt{tup}(v_a)[y_{j_b}]$. When we add t_{a+1}, t remains $\mathcal{H}$-forbidden. Further, $\texttt{tup}(v_a)$ subsumes t. Therefore there must be some $j \notin \{j_1, \ldots, j_m\}$ and $(x_i, y_j) \in E(\mathcal{H})$ such that $t_{a+1}[x_i] = t[y_j] = \texttt{tup}(v_a)[y_j] \neq \perp$. By construction, the algorithm will choose (x_i, y_j) at step (2) to expand v_a and create a child v_{a+1} that connects with an edge $(y_j, t[y_j])$. Note that, $v(t_{a+1})$ still subsumes t, is $\mathcal{H}$-forbidden for the tuples $t_1, \cdots, t_{a+1}$, and has label $\perp^*$, which proves the induction hypothesis for t_{a+1}. □

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 $\phi(\mathcal{H})$. 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.

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 Lemma 3:

Lemma 3 *$\mathcal{E}_{\mathcal{H}}(R)$ is $\mathcal{H}$-equivalent to R and has size $|\mathcal{E}_{\mathcal{H}}(R)| \leq e \cdot \phi(\mathcal{H})$.*

Proof The proof of $\mathcal{H}$-equivalence is based on the observation that if $T_{\mathcal{H}}(R) = T_{\mathcal{H}}(R')$, then R, R' must be $\mathcal{H}$-equivalent. Indeed, both trees will have the same minimally $\mathcal{H}$-forbidden tuples, and therefore the set of tuples that are $\mathcal{H}$-accepted will be same.

To see that $T_{\mathcal{H}}(R) = T_{\mathcal{H}}(\mathcal{E}_{\mathcal{H}}(R))$, consider R and suppose that we remove some tuple t that does not appear at any label of the tree (and therefore the resulting instance equals $\mathcal{E}_{\mathcal{H}}(R)$). If we keep the same order of scanned tuples when constructing both trees, the exact same tree will be produced (since t will not expand any node or add any label).

To prove the size bound, we have to give a bound on the number of nodes in the tree, $|V(T_{\mathcal{H}}(R))|$. For every possible mapping of nodes y_j to one of its neighbors in $\mathcal{H}$ (there are $\prod_{j\in[\ell]} d_{\mathcal{H}}(y_j)$ such mappings), consider the subtree of $T_{\mathcal{H}}(R)$ that contains only the paths from root to leaves where all the edges agree with the mapping (remember that each node creates a child corresponding to an edge (x_i, y_j) of $\mathcal{H}$); we will first count the nodes of such a subtree. This is because the root node can have at most ℓ children corresponding to $\leq \ell$ edges in the mapping. Each child of root can have at most $\ell - 1$ children as one of the edges in the mapping has been used in the first level. Therefore, this subtree will be of size at most

$$\ell + \ell(\ell-1) + \cdots + \ell! = \sum_{i=0}^{\ell} \frac{\ell!}{i!} = \ell! \sum_{i=0}^{\ell} \frac{1}{i!} \leq e \cdot \ell!$$

Since the union of these subtrees will cover all the nodes of $T_{\mathcal{H}}(R)$, we obtain that the $e \cdot \phi(\mathcal{H})$ is an upper bound for the size of the tree. □

Example 4 (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. Let $\mathcal{P}_q$ be 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 (input) relations and derived relations (temporary relations that are created at intermediate steps in the query plan) are distinct. We also assume, as we discussed in Section 2, that there are no inequalities of the form $x \neq c$, or $x \neq y$ where x, y are in the same atom. Such inequalities can be evaluated as selection operators at the bottom of the query plan after scanning each relation, and will incur at most an additive $O(|\mathcal{I}||D|)$ factor in the running time of the query. Thus, we can always add these selections operators after we have transformed the query plan. Our running example for this section is given below:

Example 5 Consider the query $(q_0, \mathcal{I})$, and the plan $\mathcal{P}_{q_0}$ that computes q_0:

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

The query plan $\mathcal{P}_{q_0}$ is depicted in Fig. 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 inequalities. For example, the attribute A is being projected out and it is used later in the inequality $x \neq w$ with the attribute D 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 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 att$\mathcal{P}$. If $X \subseteq \text{att}R$, let $\bar{X}^R = \text{att}R \setminus X$.

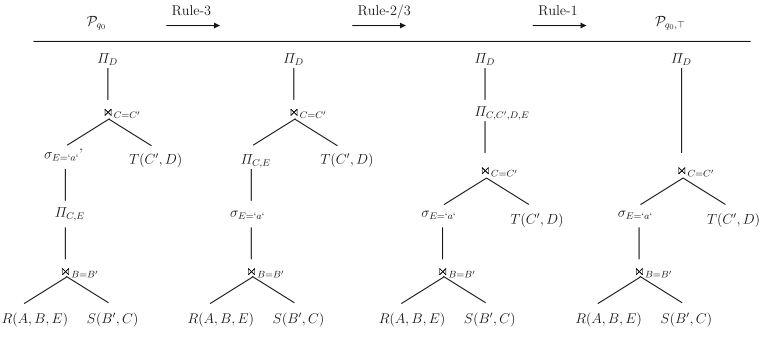


Fig. 3 The SPJ query plan $\mathcal{P}_{q_0}$ for Example 5, and the transformation to the plan $\mathcal{P}_{q_0,\top}$

Definition 6 ($\mathcal{H}$-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 $\Pi_X^{\mathcal{H}}(R)$, is defined as

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

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

The intuition behind this definition is as follows. Suppose we want to apply a projection Π_X on a base or derived relation R. In the presence of inequalities, it might be the case that some of the projected-out attributes, which are captured by $\bar{X}^R$, must be later compared with attributes that appear later in the query plan (and are in $\text{att}\mathcal{P} \setminus \text{att}R$). These comparisons are captured by the bipartite graph $\mathcal{H}$: each edge corresponds to an inequality that has to be evaluated before we output the final answer. In order to be able to evaluate the inequalities, we cannot project out the attributes as we normally do. Instead, the operator $\Pi_X^{\mathcal{H}}$ first groups the tuples from R according to the values of the X-attributes, and then for every projected value $a \in \Pi_X(R)$ computes a small $\mathcal{H}$-equivalent subrelation $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(R))$. This way, we keep all the necessary information to correctly evaluate the inequalities.

Observation 4 *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 3)*

We next present the algorithm that transforms the plan $\mathcal{P}_q$ to a plan $\mathcal{P}_{q,\mathcal{I}}$ that incorporates the inequalities.

First step 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. The key observation is 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 the original plan $\mathcal{P}_q$ 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 Fig. 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 unfortunately cannot use the standard notion of plan equivalence, since an $\mathcal{H}$-projection does not output a unique relation. Instead, we need to introduce a weaker version of plan equivalence.

Definition 7 (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 $\boldsymbol{\alpha}$, $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(\mathcal{P}_1))$ and $\mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(\mathcal{P}_2))$ 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 $\mathcal{E}_{\mathcal{H}}$. Notice of course that if two plans $\mathcal{P}_1, \mathcal{P}_2$ are equivalent under $\Pi_X^{\mathcal{H}}$, then $\Pi_X(\mathcal{P}_1) = \Pi_X(\mathcal{P}_2)$, i.e. after we apply the actual projection on X the answers will be exactly the same.

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 att$S$ and $\mathcal{H} = (\bar{X}^S, \mathbf{A}, E)$.[4] The rules are:

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

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

This rule corresponds to the absorption rule. Intuitively, it says that we can split an $\mathcal{H}$-projection on X by first applying an $\mathcal{H}'$-projection on Y, where the inequalities in $\mathcal{H}'$ are the ones that do not include any attributes in Y. In the running example,

[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 $\mathbf{A} \setminus \text{att}S$.

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 Fig. 4 (corresponding to Rule-1 in Fig. 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

$$\Pi^{\mathcal{H}}_{X_1 \cup Z_2}(\sigma_{\mathcal{I}}(R_1 \times R_2)) \equiv^{\mathcal{H}}_{X} \sigma_{\mathcal{I} \setminus \mathcal{I}[Z_1]}(\Pi^{\mathcal{H}'}_{X_1}(\sigma_{\mathcal{I}[Z_1]}(R_1)) \times R_2)$$

This rule corresponds to the distribution rule. To explain this rule in detail, consider a simple example: suppose we have the relations $R_1(A, B)$ (so $Z_1 = \{A, B\}$) and $R_2(C)$ ($Z_2 = \{C\}$), the graph $\mathcal{H}$ contains the inequality $A \neq D$. The task is to push the projection $\Pi^{\mathcal{H}}_{A,C}(\sigma_{B \neq C, A \neq B}(R_1 \times R_2))$ inside the join. The inequality $A \neq B$ includes attributes only in R_1, so we can evaluate it before the projection $\Pi_A(R_1)$. On the other hand, the inequality $B \neq C$ involves C, so when we apply the projection on R_1, we have to add to $\mathcal{H}$ the inequality $B \neq C$. The transformed plan will become $\sigma_{B \neq C}(\Pi^{\mathcal{H}'}_{A}(\sigma_{A \neq B}(R_1)) \times R_2)$, where $\mathcal{H}'$ includes now $A \neq C$, $A \neq D$.

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 Fig. 4.

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

$$\Pi^{\mathcal{H}}_{X}(\sigma_{\mathcal{I}}(\sigma_\theta(R))) \equiv^{\mathcal{H}}_{X} \sigma_\theta(\Pi^{\mathcal{H}}_{X}(\sigma_{\mathcal{I}}(R)))$$

This rule corresponds to the commutativity of selection rule. It is essentially the same as the standard rule for pushing selections inside projections. In the running

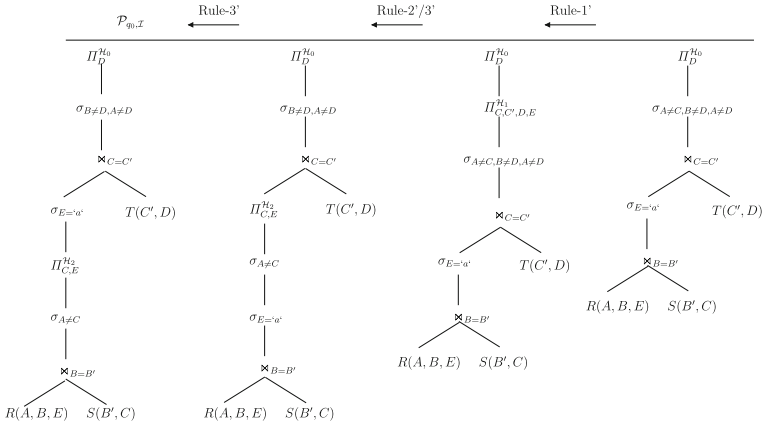


Fig. 4 The reverse application of rules for Example 5. The bipartite graphs defined have edge sets $E(\mathcal{H}_0) = \emptyset$, $E(\mathcal{H}_1) = \emptyset$ and $E(\mathcal{H}_2) = \{(A, D), (B, D)\}$

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

Lemma 4 *The three rules (Rule-1'), (Rule-2'), (Rule-3') preserve the equivalence of the plans under* $\Pi^{\mathcal{H}}_X$.

Proof We show the equivalence for each rule. For convenience, we will use the notation $\mathcal{I} \models t$ to mean that the inequalities in the set $\mathcal{I}$ are satisfied by the values of the tuple t.

(Rule-1'). Denote $S' = \sigma_{\mathcal{I}}(S)$. It suffices to show that for every tuple $\boldsymbol{\alpha}$, $\mathcal{E}_1 = \mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(S'))$ and $\mathcal{E}_2 = \mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(\Pi^{\mathcal{H}'}_Y(S')))$ are $\mathcal{H}$-equivalent. Let us fix some $X = \boldsymbol{\alpha}$.

The one direction is based on the observation that $\Pi^{\mathcal{H}'}_Y(S') \subseteq S'$. Hence, $\sigma_{X=\boldsymbol{\alpha}}(\Pi^{\mathcal{H}'}_Y(S')) \subseteq \sigma_{X=\boldsymbol{\alpha}}(S')$, which implies that if a tuple is $\mathcal{H}$-accepted by $\mathcal{E}_2$, it is accepted by $\mathcal{E}_1$ as well.

For the other direction, suppose that t is $\mathcal{H}$-accepted by $\mathcal{E}_1$. Then, there exists some $s \in \sigma_{X=\boldsymbol{\alpha}}(S')$ such that $E \models s \circ t$.[5] Since $\bar{Y}^S \subseteq \bar{X}^S$, $E[\bar{Y}^S, A] \models s \circ t$ and t must be $\mathcal{H}'$-accepted by $\sigma_{Y=s[Y]}(\sigma_{X=\boldsymbol{\alpha}}(S'))$, and consequently by $\mathcal{E}_{\mathcal{H}'}(\sigma_{Y=s[Y]}(\sigma_{X=\boldsymbol{\alpha}}(S')))$ as well. Then, there exists some $s' \in \mathcal{E}_{\mathcal{H}'}(\sigma_{Y=s[Y]}(\sigma_{X=\boldsymbol{\alpha}}(S')))$ such that $E[\bar{Y}^S, A] \models t \circ s'$. However, since $s'[Y] = s[Y]$, we must also have that $E \models t \circ s'$. Since $s' \in \Pi^{\mathcal{H}'}_Y(\sigma_{X=\boldsymbol{\alpha}}(S'))$, we conclude that t is $\mathcal{H}$-accepted by $\mathcal{E}_2$.

(Rule-2'). Denote $R'_1 = \sigma_{\mathcal{I}[Z_1]}(R_1)$ and $\mathcal{I}_1 = \mathcal{I} \setminus \mathcal{I}[Z_1]$. It suffices to show that for every tuple $\boldsymbol{\alpha}$, the following are $\mathcal{H}$-equivalent:

$$\mathcal{E}_1 = \mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(\sigma_{\mathcal{I}'}(R'_1 \times R_2))),$$
$$\mathcal{E}_2 = \mathcal{E}_{\mathcal{H}}(\sigma_{X=\boldsymbol{\alpha}}(\sigma_{\mathcal{I}'}(\Pi^{\mathcal{H}_1}_{X_1}(R'_1) \times R_2)))$$

The one direction of the equivalence is based on the fact that $\Pi^{\mathcal{H}_1}_{X_1}(R'_1) \subseteq R'_1$. The other direction is more involved.

Suppose that t is $\mathcal{H}$-accepted by $\mathcal{E}_1$. Then, there exists some $s \in R'_1 \times R_2$ such that $E, \mathcal{I}' \models s \circ t$ and $s[X] = \boldsymbol{\alpha}$. Now, consider the tuple $t \circ s[Z_2]$. The crucial observation is that $t \circ s[Z_2]$ is $\mathcal{H}_1$-accepted by $\sigma_{X_1=\boldsymbol{\alpha}[X_1]}(R'_1)$, and thus by $\mathcal{E}_{\mathcal{H}_1}(\sigma_{X_1=\boldsymbol{\alpha}[X_1]}(R'_1))$ as well. Then, there exists some $s_1 \in \mathcal{E}_{\mathcal{H}_1}(\sigma_{X_1=\boldsymbol{\alpha}[X_1]}(R'_1))$ such that $E \models t \circ s_1 \circ s[Z_2]$. Finally, observe that the tuple $s' = s_1 \circ s[Z_2]$ belongs in $\Pi^{\mathcal{H}_1}_{X_1}(R'_1) \times R_2$, has $s'[X] = \boldsymbol{\alpha}$, and also satisfies $\mathcal{I}'$. This implies that t is $\mathcal{H}$-accepted by $\mathcal{E}_2$.

(Rule-3'). This is immediate, since the selection θ is applied only on the attributes in X, which are not projected out.

□

[5] $s \circ t$ denotes the concatenation of s, t.

After applying the above transformations in the reverse order, we obtain a plan $\mathcal{P}_{q,\mathcal{I}}$ where each relational operator is in the same position in the query plan as in the original plan $\mathcal{P}_q$, plus some additional inequality conditions. More formally, the following lemma holds:

Lemma 5 *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 $\Pi^{\mathcal{H}}_X$ operator in $\mathcal{P}_{q,\mathcal{I}}$ for some appropriately constructed $\mathcal{H}$.*
3. *Every derived 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 derived relation in $\mathcal{P}_q$.*
4. *If $T(|q|, |D|)$ is the time to evaluate $\mathcal{P}_q$, 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. 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 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 6 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.

Computing CQs with non-SPJ Plans So far 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 [20, 23] 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 non-standard algorithms for Boolean CQs run faster than SPJ algorithms. For example, the *cycle query*

$$C_{2k}() = R(x_1, x_2), R(x_2, x_3), \ldots, R(x_{2k}, x_1)$$

for any $k \geq 1$ can be computed in time $O(N^{2-1/k})$, where $N = |R|$. We can show that we can apply our techniques in this case as well, even though it remains open whether we can use them for any black-box algorithm.

We first present the algorithm that computes C_{2k}.

Theorem 5 *C_{2k} can be computed in time $O(N^{2-1/k})$, where $N = |R|$.*

Proof Let δ be some threshold parameter. We say that a value a is a *heavy hitter* if the degree $|\sigma_{X=a} R(X, Y)| \geq \delta$, otherwise it is light. The algorithm distinguishes two cases.

First, we compute all the $2k$-cycles that contain some heavy hitter value. We have at most N/δ such values. For each such value, we can compute

$$C_{2k}^{(a)}() = R(a, x_2), R(x_2, x_3), \ldots, R(x_{2k}, a)$$

Observe that this is an acyclic query now that a is a fixed value, so we can compute this query in time $O(kN)$. Hence, to compute all possible cycles in this case we need $O(kN^2/\delta)$ time.

Second, we compute whether there exists a cycle C_{2k} that uses only light values. To do this, let R' be the subset of R that contains only the light values. The maximum degree is δ, so the queries

$$\begin{aligned} q_1(x_1, x_k) &= R(x_1, x_2), \ldots, R(x_{k-1}, x_k) \\ q_2(x_k, x_1) &= R(x_k, x_{k+1}), \ldots, R(x_{2k}, x_1) \end{aligned}$$

each contain at most $N\delta^{k-1}$ answers, which we can compute in time $O(kN\delta^{k-1})$ by performing consecutive joins. However, $|q_1|, |q_2|$ have size at most N, and we can compute their intersection in time $O(N \log N)$. So the total running time for this case is $O(N\delta^{k-1})$.

To balance the two cases, we must have $N^2/\delta = N\delta^{k-1}$ or $\delta = N^{1/k}$. □

We can combine the above algorithm with our technique as follows. Suppose the query now is $(C_{2k}, \mathcal{I})$ for some set of inequalities $\mathcal{I}$. Observe that the first case is easy to handle, since we know how to compute $C_{2k}^{(a)}$ in time $O(kN \times \max_{\mathcal{H}}(\phi(\mathcal{H}))^2)$, where $\phi(\mathcal{H})$ depends only on the inequality structure. For the second case, instead of computing q_1, q_2, we consider the full queries

$$\begin{aligned} q_1^f(x_1, x_2, \ldots, x_k) &= R(x_1, x_2), \ldots, R(x_{k-1}, x_k) \\ q_2^f(x_k, \ldots, x_{2k}, x_1) &= R(x_k, x_{k+1}), \ldots, R(x_{2k}, x_1) \end{aligned}$$

and compute $(q_1^f, \mathcal{I}_1)$, $(q_2^f, \mathcal{I}_2)$, where $\mathcal{I}_1$ are the inequalities defined only between the variables of q_1 (and similarly for $\mathcal{I}_2$). Since these queries have size at most $N\delta^{k-1}$, we can compute the full answers and apply the inequalities at the end. To compute the intersection between q_1, q_2, let $\mathcal{I}_{12} = \mathcal{I} \setminus (\mathcal{I}_1 \cup \mathcal{I}_2)$. We then compute $\Pi_{x_1,x_k}^{\mathcal{H}_1}(q_1^f)$, where $\mathcal{H}_1 = (\{x_2, \ldots, x_{k-1}\}, \{x_{k+1}, \ldots, x_{2k}\}, \mathcal{I}_{12})$, and similarly $\Pi_{x_1,x_k}^{\mathcal{H}_2}(q_2^f)$ with a symmetrically defined $\mathcal{H}_2$. The resulting projections have size at most $N \cdot \phi(\mathcal{H}_i)$

for $i = 1, 2$, so we can then compute their intersection in time $O(N \log N)$ and then apply the inequalities in $\mathcal{I}_{12}$.

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).[6]

Theorem 6 *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 $\mathcal{I}$.*

First, we state the original randomized color-coding technique to describe the intuition. Let h be a hash function that maps each value of the active domain Dom of the instance D to a random *color*. The basic idea of the color-coding technique is to randomly color each value of the active domain Dom by using h, use these colors to check the inequality constraints, and use the actual values of the attributes of the tuples for checking the equality constraints.

Definition 8 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 $h : \texttt{Dom} \to [p]$[7] (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 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

[6] The $\log^2(|D|)$ factor in Theorem 1 is reduced to $\log(|D|)$ in Theorem 6, but this is because one log factor was due to sorting the relations in the acyclic query, and now this hidden in the term $T(|q|, |D|)$.

[7] We use $[p]$ to denote the set $\{1, \ldots, p\}$.

$|\mathcal{F}| = 2^{O(k)} \log(|\mathtt{Dom}|) = 2^{O(k)} \log |D|$ [8], and evaluating each hash value takes only $O(1)$ time.

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 $\bigcup_{h\in\mathcal{F}} \bigcup_{\mathbf{c}\in\mathcal{C}(G^{\mathcal{I}})} q(D[\mathbf{c}, h])$.*

Proof of Theorem 6 Suppose

$$q^{(h)}(D) = \{t[head(q)] \mid t \in q^f(D) \models_h \mathcal{I}\}$$

Then the union $\bigcup_{h\in\mathcal{F}} q^{(h)}(D)$ produces the result of the query (this is because for any tuple $t \in q^f(D)$, there exists a hash function $h \in \mathcal{F}$ that satisfies all the inequalities in $\mathcal{I}$). In the rest of this subsection, we will show how to compute $q^{(h)}(D)$ for a fixed hash function $h : \mathtt{Dom} \to [p]$, $p \geq k$, using the coloring technique in time bounded by $2^{O(k \log k)} T(|q|, |D|)$.

Let $\mathbf{C}$ be a *valid p-coloring* of the vertices of the inequality graph $G^{\mathcal{I}}$, such that whenever $x_i \neq x_j$, we have that $c_i \neq c_j$ where c_i denotes the color of variable x_i under $\mathbf{c}$. For each such coloring, 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.

Lemma 6 *Let $\mathcal{C}(G^{\mathcal{I}})$ denote all the valid colorings of $G^{\mathcal{I}}$. Then,*

$$q^{(h)}(D) = \bigcup_{\mathbf{C}\in\mathcal{C}(G^{\mathcal{I}})} q(D[\mathbf{C}, h])$$

Proof Let $t \in q^f(D) \models_h \mathcal{I}$. Let $\mathbf{C}$ be the coloring such that for every $x_i \in V(G^{\mathcal{I}})$, we set $c_i = h(t[x_i])$. We will show that $\mathbf{C}$ is a valid coloring of $G^{\mathcal{I}}$. Indeed, if $x_i \neq x_j \in \mathcal{I}$, it must be that $h(t[x_i]) \neq h([t_j])$ (Since $t \models_h \mathcal{I}$) and hence $c_i \neq c_j$. Thus, we have that $t[head(q)] \in q(D[\mathbf{C}, h])$.

For the other direction, let $t \in q^f(D[\mathbf{C}, h])$ for a valid coloring $\mathbf{C}$. For any inequality $x_i \neq x_j$, we will have $h(t[x_i]) = c_i \neq c_j = h(t[x_j])$, and hence $t \models_h \mathcal{I}$. □

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

The algorithm now iterates over all hash functions $h \in \mathcal{F}$, and all valid colorings of $G^{\mathcal{I}}$ with p colors, and for each combination computes $q(D[\mathbf{c}], h)$. The output result is:

$$\bigcup_{h \in \mathcal{F}, \mathbf{C} \in \mathcal{C}(G^{\mathcal{I}})} q(D[\mathbf{C}, h])$$

The running time is $O(|\mathcal{F}| \cdot |\mathcal{C}(G^{\mathcal{I}})| \cdot T(q, |D|))$. As we discussed before $|\mathcal{F}| \leq 2^{O(p)} \log |D|$ and $|\mathcal{C}(G^{\mathcal{I}})| \leq k^p$. Theorem 6 follows by choosing $p = k = |V(G^{\mathcal{I}})|$ (the smallest possible value of p). □

Comparison of Theorem 2 with Theorem 6 The factors dependent on the query in these two theorems ($g(q, \mathcal{I})$ in Theorem 2 and $f(k)$ in Theorem 6) 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. Further, apart from the additional $\log(|D|)$ factor, the derandomized color-coding technique demands the construction of a different k-perfect hash family for every database instance and every query issued on the instance (since the hash family construction depends on the active domain and the number of variables in the 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 a more sophisticated projection operation.

More importantly, the color coding technique is *oblivious* of the combined structure of the query and the inequalities (as it was originally used for the complete inequality graph), and has exponential dependency in k even for very simple inequality patterns. In particular, our algorithm can compute certain queries with polynomial combined complexity, whereas color-coding leads to exponential running time in k. 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 [6, 24]. We give examples of such q and $\mathcal{I}$ below:

Example 7 Consider the path query $P^k(\ )$, which is acyclic, and consider the following inequality patterns (see Fig. 5):

1. $(P^k, \mathcal{I}_1)$ where $\mathcal{I}_1 = \{x_i \neq x_{i+2} : i \in [k-1]\}$ has treewidth 2. A tree-decomposition with treewidth 2 (ie, maximum node in the tree has size 3) is $\{x_i, x_{i+1}, x_{i+2}\} — \{x_{i+1}, x_{i+2}, x_{i+3}\} — \cdots$.

2. $(P^k, \mathcal{I}_2)$ where $\mathcal{I}_2 = \{x_i \neq x_{i+\frac{k}{2}} : i \in [\frac{k+1}{2}]\}$ has treewidth 3 (k is odd): The incidence graph without the edge $(x_{\frac{k}{2}}, x_{\frac{k}{2}} + 1)$ has the structure of a $\frac{k}{2} \times 2$ grid and therefore has treewidth 2 (see Fig. 5b). We can simply add the node $x_{\frac{k}{2}}$ to all nodes in this tree-decomposition to have a decomposition with treewidth 3.
3. $(P^k, \mathcal{I}_3)$ where $\mathcal{I}_3 = \{x_i \neq x_{k-i+1} : i \in [\frac{k+1}{2}]\}$ has treewidth 2 (k is odd): A tree-decomposition with treewidth 3 can be obtained by going back and forth along the inequality (dotted) edges: $\{x_{i+1}, x_i, x_{k-i+1}\} — \{x_{k-i+1}, x_{k-i}, x_{i+1}\} — \cdots$. For example, in Fig. 5c the decomposition can be $\{x_2, x_1, x_8\} — \{x_8, x_7, x_2\} — \{x_3, x_2, x_7\} — \{x_7, x_6, x_3\} — \{x_4, x_3, x_6\} — \{x_6, x_5, x_4\}$.

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 Fig. 5a), the techniques of [24] 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 8 Consider $(P^k, \mathcal{I}_4)$ (see Fig. 5d), where $k+1 = p^2$ for some p. Algebraically, we can write $\mathcal{I}_4$ as: $\mathcal{I}_4 = \{x_i \neq x_{\lfloor i/p \rfloor + 1 + 2p - (i \mod p)} | i = 1, \ldots, p(p-1)\}$. 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

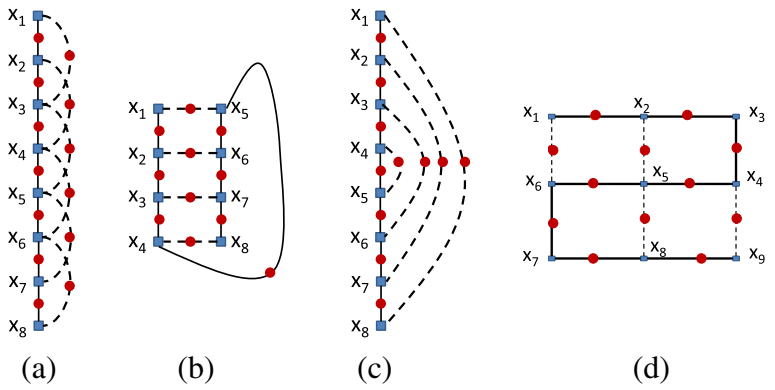


Fig. 5 Augmented graphs for Example 7 ($k = 7$) and Example 8 ($k = 8$). The solid and dotted edges come from the query and inequalities respectively; the blue squares denote variables, and red circles denote (unnamed) relational atoms: (**a**) $(P^7, \mathcal{I}_1)$, (**b**) $(P^7, \mathcal{I}_2)$,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

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 9 (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 1 *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 (treewidth 1).*

Proof We reduce from list coloring on grid graphs, which is known to be NP-complete with $c = 4$ colors and where $2 \leq |L(v)| \leq 3$ for each vertex v [7].

Given an instance of the list coloring problem where the graph G is a $p \times p$ grid-graph, we create an instance of $P^k, \mathcal{I}_4$ as shown in Fig. 5d, where $k + 1 = p^2$. We denote by x_i both the vertices in G as welll as the variables in P^k. For each $i \in [k]$, we create an instance

$$R_i(x_i, x_{i+1}) = \{(a, b) \ : \ a \neq b \text{ and } a, b \in L(x_i) \times L(x_j)\}$$

The inequalities $\mathcal{I}_4$ are as shown in the figure: $x_i \neq x_j$. Note that each vertex v in the grid graph G appears in one of the relations so its domain in the query is bounded by $L(v)$.

Suppose the list coloring instance has a valid coloring, i.e., every vertex v in G can be colored $c[v] \in L(v)$ such that for each edge (u, v), $c[u] \neq c[v]$. This gives an yes-instance to the query $(P^k, \mathcal{I}_4)$. Similarly, if the query has a yes instance, that corresponds to a yes-instance of the list coloring problem. □

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 2 *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 7). However, list coloring can be solved in polynomial time for certain graphs $G^{\mathcal{I}}$:

- **Trees** (the problem can be solved in time $O(|V|)$ independent of the available colors[13]), and in general graphs of constant treewidth.
- **Complete graphs** (by a reduction to *bipartite matching*).[9]

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 3 *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.*

Proof Let $X = \{x_{i_1}, \ldots, x_{i_c}\}$ be the vertices of the vertex cover. Consider each possible instantiation of these variables from the domain $\mathtt{Dom}$; the number of such instantiation is $|\mathtt{Dom}|^c$. For each such instantiation α consider the updated query q^α. Since X is a vertex cover and each relation has arity ≤ 2, in q^α each relation has at most one free variable. Relations with single variable that has been instantiated to a unique constant from α or relations where both the variables have been instantiated can be evaluated by a linear scan of the instance and removed thereafter. Similarly, relations with arity 2 where exactly one of the two variables has been instantiated to a constant can be evaluated by removing tuples from the instance that are not consistent with this constant. These steps can be done in poly-time in combined complexity. In the reduced query, each relation has exactly one free variable and therefore is equivalent to F^n for some n ($n =$ the number of relations in the query where exactly one variable belongs to X). Hence if list coloring can be solved in poly-time on $G^{\mathcal{I}}$, $(q^\alpha, \mathcal{I})$ for each instantiation α, and therefore $(q, \mathcal{I})$ can be solved in poly-time in combined complexity. □

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 $\alpha \in \mathtt{Dom}$, the query $R_1(\alpha, x_1), \ldots, R_n(\alpha, x_n)$ is equivalent to F^n,

[9] 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.

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 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 10 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 $\leq b$.

The class of 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.

The main theorem of this section is stated below:

Theorem 7 *The following hold:*

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 the combined complexity of $(q, \mathcal{I})$ is poly-time for $q \in \mathcal{Q}$.*

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 [5, 12] 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.

Proof **1. NP-hardness.** We do a reduction from 3-COLORING. Let $G = (V, E)$ be an undirected graph, where $n = |V|$. The goal is to color G with 3 colors such that no two adjacent vertices have the same color. Given the family $\mathcal{Q}$ with unbounded

integer vertex packing, we construct a query with inequality $(q, \mathcal{I})$ where $q \in \mathcal{Q}$, and an instance D such that G admits a 3-coloring if and only if $(q, \mathcal{I})$ is true on the instance D.

Find in polynomial time the query $q \in \mathcal{Q}$ such that q has an integer vertex packing X of size n. Let $X = \{x_1, \ldots, x_n\}$ be the variables in the vertex packing. We create an instance D as follows. Note that each relation R contains at most one variable x_i from X. If R contains no such variable, then $R^D = \{(0, 0, \ldots, 0)\}$ (a single tuple with value 0 for all variables). Otherwise, let $x_i \in vars(R)$ and without loss of generality (wlog.), let x_i be at the first position of R; then, $R^D = \{(c, 0, 0, \ldots) \mid c = 1, 2, 3\}$. Observe that the size of the instance D is at most $3 \cdot |q|$ and it is constructed in poly-time.

By construction the answer to the full query q^f of q is $q^f(D) = \{1, 2, 3\}^n \times \{(0, 0, \ldots)\}$, where wlog. all $x_1, \ldots, x_n$ appear at the first n positions of the head of q^f. Therefore, each variable x_i, $i = 1, \ldots, n$ can obtain each color independent of the other variables. Finally, we construct a one-to-one mapping from each vertex $v \in V$ to a unique variable $x_v \in \{x_1, \ldots, x_n\}$, and define $\mathcal{I} = \{x_u \neq x_v \mid (u, v) \in E\}$. Now it is easy to verify that G has a valid 3-coloring if and only if $(q, \mathcal{I})$ is true on D.

2. Algorithm for queries with bounded fractional vertex packing. Since the maximum fractional vertex packing of any $q \in \mathcal{Q}$ is $\leq b$, the minimum fractional edge cover is also $\leq b$ by duality. Thus from [5, 12], $q^f(D)$ can be evaluated in poly-time in combined complexity (in time $O(|q|^2|D|^{b+1})$). Further, $|q^f(D)| \leq |D|^b$ [5, 12]. We first compute $q^f(D)$, then for each tuple in $q^f(D)$ we check whether it satisfies the inequalities, and finally apply the projection to get the answers to $(q, \mathcal{I})$ in poly-time in combined complexity.

□

We illustrate the properties with some 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 $\mathcal{I}$. More generally, observe that any family of queries with a bounded number of atoms (but unbounded number of variables) has a bounded vertex packing. 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 2).

Theorem 7 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.[10]

[10]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? In particular extending our techniques to UCQs seems plausible, since we can consider SPJU plans (where union is added as a relational operator) and notice that projection distributes over union; the analogue of this transformation rule in the inequality setting would allow us to carry over our results to UCQs. Can our techniques be used as a black-box to extend any algorithm for CQs, ie, not necessarily based on SPJ query plans, to evaluate CQs with inequalities?

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases Addison-Wesley (1995)
2. Afrati, F., Li, C., Mitra, P.: Answering Queries Using Views with Arithmetic Comparisons. In: PODS, pp. 209–220 (2002)
3. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997)
4. Alon, N., Yuster, R., Zwick, U.: Color Coding. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms. Springer (2008)
5. Atserias, A., Grohe, M., Marx, D.: Size bounds and query plans for relational joins. FOCS 739–748 (2008)
6. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. Theor. Comput. Sci **239**(2), 211–229 (2000)
7. Demange, M., De Werra, D.: On some coloring problems in grids. Theor. Comput. Sci **472**, 9–27 (2013)
8. Durand, A., Grandjean, E.: The complexity of acyclic conjunctive queries revisited coRR abs/cs/0605008 (2006)
9. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. J. ACM **49**(6), 716–752 (2002)
10. Gottlob, G., Leone, N., Scarcello, F.: Hypertree Decompositions and Tractable Queries. In: PODS, pp. 21–32 (1999)
11. Graham, M.: On the Universal Relation. Technical Report. University of Toronto, Ontario (1979)
12. Grohe, M., Marx, D.: Constraint Solving via Fractional Edge Covers. In: SODA, pp. 289–298 (2006)
13. Jansen, K., Scheffler, P.: Generalized coloring for tree-like graphs. Discret. Appl. Math. **75**(2), 135–155 (1997)
14. Khayyat, Z., Lucia, W., Singh, M., Ouzzani, M., Papotti, P., Quiané-Ruiz, J., Tang, N., Kalnis, P.: Lightning fast and space efficient inequality joins. PVLDB **8**(13), 2074–2085 (2015). http://www.vldb.org/pvldb/vol8/p2074-khayyat.pdf
15. Klug, A.: On conjunctive queries containing inequalities. J. ACM **35**(1), 146–160 (1988)
16. Kolaitis, P.G., Martin, D.L., Thakur, M.N.: On the Complexity of the Containment Problem for Conjunctive Queries with Built-In Predicates. In: PODS, pp. 197–204 (1998)
17. Koutris, P., Milo, T., Roy, S., Suciu, D.: Answering Conjunctive Queries with Inequalities. In: ICDT, pp. 76–93 (2015)
18. van der Meyden, R.: The complexity of querying indefinite data about linearly ordered domains. J. Comput. Syst. Sci **54**(1), 113–135 (1997). doi:10.1006/jcss.1997.1455
19. Monien, B.: How to Find Long Paths Efficiently. In: Ausiello, G., Lucertini, M. (eds.) Analysis and Design of Algorithms for Combinatorial Problems, North-Holland Mathematics Studies, vol. 109, pp. 239–254. North-Holland (1985)

20. Ngo, H.Q., Porat, E., Ré, C., Rudra, A.: Worst-Case Optimal Join Algorithms: [Extended Abstract]. In: PODS, pp. 37–48 (2012)
21. Papadimitriou, C.H., Yannakakis, M.: On the Complexity of Database Queries. In: PODS, pp. 12–19 (1997)
22. Robertson, N., Seymour, P.: Graph minors. iii. planar tree-width. J. Comb. Theory B **36**(1), 49–64 (1984)
23. Veldhuizen, T.L.: Triejoin: a Simple, Worst-Case Optimal Join Algorithm. In: ICDT, pp. 96–106 (2014)
24. Yannakakis, M.: Algorithms for Acyclic Database Schemes. In: VLDB, pp. 82–94 (1981)
25. Yu, C., Ozsoyoglu, M.Z.: An Algorithm for Tree-Query Membership of a Distributed Query. In: COMPSAC, pp. 306–312 (1979)
26. Yuster, R., Zwick, U.: Finding even cycles even faster. SIAM J. Discrete Math. **10**(2), 209–222 (1997)