On the Complexity of Database Queries

(Extended Abstract)

Christos H. Papadimitriou
Division of Computer Science, U. C. Berkeley, Berkeley, CA 94720

Mihalis Yannakakis
Bell Laboratories, Lucent Technologies, Murray Hill NJ 07974

Abstract

We revisit the issue of the complexity of database queries, in the light of the recent *parametric* refinement of complexity theory. We show that, if the number of variables in the query (or the query size) is considered as a parameter, the familiar query languages (conjunctive, positive, first order, Datalog) are classified at appropriate levels of the so-called W hierarchy of Downey and Fellows. These results strongly suggest that the query size is inherently in the exponent of the data complexity of any query evaluation algorithm, with the implication becoming stronger as the expressibility of the query language increases. On the positive side, we show that this exponential dependence can be avoided for the extension of acyclic queries with $\neq$ (but not $<$) inequalities.

1 Introduction

The complexity of query languages has been —next to expressibility— one of the main preoccupations of database theory ever since the paper by Chandra and Merlin twenty years ago [4]; see [6, 1] for extensive overviews of the subject. It has been noted rather early [14] that, when considering the complexity of evaluating a query on an instance, one has to distinguish between two kinds of complexity: *Data complexity* is the complexity of evaluating a query on a database instance, *when the query is fixed,* and we express the complexity as a function of the size of the database. The other, called *combined complexity,* considers both the query and the databse instance as input variables; the combined complexity of a query language is typically one exponential higher than data complexity.[1] Of the two, data complexity is widely regarded as more meaningful and relevant to database research, since

[1]A third kind, *expression complexity* assumes that the database instance is fixed, and is rarely differentiated from the combined complexity.

the query is typically of a size that can be productively assumed to be fixed, and is in any event much smaller than a typical database.

For a broad range of important query languages (relational laguages like conjunctive queries, first-order, Datalog, fixpoint logic, as well as constraint languages, i.e., extensions with constraints such as arithmetic comparisons, linear and polynomial inequalities etc.) data complexity predicts that the query evaluation problem is perfectly tractable: the complexity classes spanned by these query languages range from AC_0 to P, well within what is considered satisfactory in complexity theory. These tractability results are often quoted in the literature to suggest that the corresponding computational problems are tractable, well-understood, solved, under control. This implication is based on the thesis, broadly accepted in the theory of algorithms, that, as a rule, polynomial algorithms that arise in practice are usually fast, practical, with tolerable constant coefficient and reasonable exponents. Is this conclusion justified in the context of database query processing?

It seems to us that neither of the two notions of complexity is completely satisfactory. On the one hand, combined complexity is rather restrictive because it treats queries and databases as part of the input the same way, even though the size q of queries is typically orders of magnitude smaller than the size n of the database. Indeed it is for this reason that the study of the complexity of query languages has mostly concentrated on data complexity. However, on the other hand, polynomial time in the context of data complexity means time n^q, and in fact the known algorithms that place the above mentioned languages in P have precisely such a running time. Even though $q << n$, it is not reasonable to consider q fixed, because even for small values of q, a running time of n^q hardly qualifies as tractable, especially in view of the fact that n is typically huge. What should the notion of complexity be then? What we would like to have is a running time in which n is not raised to a power that depends on q, i.e. the dependence on n is of the form n^c where c is a constant independent of the query (and hopefully very small).

Let us draw an analogy with the computer-aided verification area. The basic problem there is the model checking problem: does a given program P (the 'model') satisfy a desired property ϕ (expressed in some specification language such as LTL, propositional linear temporal logic). There have been significant advances in recent years in the development of algorithms and tools in this area, especially for finite-state programs, which cover an important set of critical applications. The model checking problem for finite state programs P and LTL specifications ϕ is PSPACE-complete. However, usually specifications are rather small (like queries) and programs are quite large (like databases). Fortunately, it turns out that the model checking problem for LTL specification ϕ and program P can be solved in time exponential in $|\phi|$ and linear in $|P|$ [9]. Can we hope for such algorithms in the query evaluation of the important query languages (such as the ones mentioned above)? What are natural classes of queries that possess this type of algorithms?

Parametric complexity provides the framework to examine these problems. We now know that there is a class of reasonably natural problems that do not fall into this mold: *parametric problems,* such as "does graph G have a clique of size k?" This problem, like many others like it, is currently solvable only by algorithms of complexity n^k. Query evaluation problems lie ominously within the scope of this category, with query length being the obvious analog of k in the parametric clique problem above. Researchers in complexity have recently developed a theory of *limited nondeterminism* and *fixed-parameter tractability* [3, 11, 5] which seeks to make important distinctions, along the lines suggested above, between problems below NP.

In particular, parametric problems with input, say, (G, k) which are solvable in polynomial time when k is fixed, can be subdivided into two broad categories: Those for which the polynomial is of the form $n^{f(k)}$ —i.e., "has k in the exponent"— and those for which it is of the form $g(k)n^c$ for some constant c. It is of great interest to distinguish between these two categories, and to develop rigorous tools that classify problems with respect to them. Downey and Fellows have introduced a sequence of complexity classes of parametric problems, collectively called *the W hierarchy*, which capture reasonably well this important issue [5]. The classes of the W hierarchy are indexed by the numbers $1, 2, \ldots$, plus two limiting classes W[SAT] and W[P]. These classes are quite rich in complete problems; the higher the W class, the less likely that the problem has a polynomial algorithm with time bound of the form $g(k)n^c$.

The main point of this paper is that *parametric complexity theory is a productive framework for studying the complexity of query languages.* In particular, our goal is to put the well-known tractability results of the query languages mentioned above under a different perspective, which renders them perhaps less confusing and misleading. In particular, we prove that the parametric versions of the query evaluation problem for conjunctive queries, positive queries, first-order queries, and Datalog queries, are hard for higher and higher levels of the W hierarchy. Therefore, it is likely that any algorithm for the corresponding query languages must have the parameter inherently in the exponent; furthermore, this likelihood increases measurably with the expressibility of the language.

We analyse the complexity for two types of parameters: the query size q and the number of variables v that appear in the query. The latter parameter is motivated by recent work of Vardi [15], who studied the complexity of queries assuming that the number of variables v is fixed, while the size of the query can grow along with the database. He found that this assumption brings the combined complexity closer to data complexity, namely polynomial time for the above languages, although the polynomial now has v in the exponent of n instead of q. Our analysis for the two parameters yields generally similar results (with some subtle differences).

In the next section we give the necessary definitions from the (evolving) field of parametric complexity. In Section 3 we give the necessary definitions for applying this theory to query problems. In Section 4 we prove our classification results. Finally, in Section 5 we show a parametric tractability result which generalizes the main tractability result known so far in database theory, namely, that acyclic queries can be evaluated efficiently (even with respect to combined complexity). We show that acyclic conjunctive queries extended *with inequalities* (conjuncts of the form $x \neq y$) are parametrically tractable, in that they can be evaluated in time almost linear in the size of the database and the output, and exponential in the size of the query or the number of variables (this exponential dependence on the parameter is unavoidable, as the inequalities turn the combined complexity of the problem from polynomial to NP-complete). Trying to extend this further to $<$ constraints leads however to parametric hardness.

2 Parametric Complexity Theory

We introduce next the main concepts from the complexity theory of parametric problems. Our definitions generally follow [5]. A *parametric problem* is a set L of pairs (x, k), where x is a string and k an integer parameter. A parametric problem is called *fixed parameter (f.p.) tractable* if there is an algorithm A that determines whether $(x, k) \in L$ in time bounded by a function of the form $f(k) \cdot |x|^c$ for some constant c; we will say that A runs in f.p. polynomial time.

Several NP-complete problems when supplied with a meaningful, natural parameter yield parametric problems that are f.p. tractable. Examples: Given a graph and k pairs of nodes, are there node-disjoint paths between all pairs of nodes? [12] Given a graph and an integer k, is there a path of length k in the graph? [10, 2] Both problems, and many others like them, have algorithms with running time $f(k) \cdot n^c$, where n is the input size and c a constant.

In contrast, several other NP-complete problems do not seem to be tractable when considered as parametric problems with the natural parameter; examples include important problems such as clique, dominating set, bandwidth, etc. All these problems are solvable in time growing as $O(n^k)$ or a similar function, where n is the input length and k the parameter (desired clique size, dominating set size, and bandwidth size in the three examples above), and, despite considerable effort to this end, no algorithm for each one of them is known with running time without k appearing in the exponent.

It would be very interesting to develop a refinement of NP-completeness theory that anticipates this sophisticated form of apparent intrectability. Such a theory has been emerging from the work of many people, but most recently and notably Downey and Fellows [5]. There appears to be a *hierarchy* of parametric problems, called the *W hierarchy*, which classifies many of these problems. We first need to introduce an appropriate notion of reduction (in the literature one finds several more general kinds of reductions, but the one given next turns out to be the more useful one, certainly for the purposes of this paper).

A *parametric reduction* between two parametric problems A and B is an algorithm which solves any instance (x, k) of A using the answers to several instances (y_i, ℓ_i) of B, where (1) all ℓ_i are upper bounded by $g(k)$ (independent of x) for some function g, and (2) the instances of B and the final answer can be constructed in time $h(k)|x|^s$, for some function h and integer s. Such reductions are often *parametric transformations*, producing for any instance (x, k) of A an equivalent instance (y, ℓ) of B, and running in time $h(k)|x|^s$ for some recursive function h and integer s.

Consider a Boolean circuit with AND, OR, and NOT gates and one output. We allow OR and AND gates of unbounded fan-in. The *depth* of a circuit is the longest path from any input to the output. Let us now define `depth-`t `weighted satisfiability` for $t > 1$, to be the following parametric problem: Given a depth-t circuit C and an integer k, is there a setting of the inputs of C *with k inputs set to 1* so that the output of C is 1? For $t = 1$ we require that the given circuit C be a 3-CNF formula. Also, the (unrestricted) `weighted circuit satisfiability` is the same problem with no depth restriction: Given a circuit C and an integer k, is there a setting of the inputs of C with k inputs set to 1, so that the output of C is 1? Finally, the `weighted formula satisfiability` problem is the case where the circuit has fan-out 1 (i.e. it is a Boolean formula).

We are now ready to define the classes in the W hierarchy; we give the definition in terms of their complete problems. We define W[t] to be the set of all parametric problems that reduce to `depth-`t `weighted satisfiability`. The limiting classes W[SAT] and W[P], are the sets of all parametric problems that reduce respectively to `weighted formula` and `weighted circuit satisfiability`, with unlimited depth. In [5] it is pointed out that these classes have many natural complete problems, under parametric reductions. For example, `clique` is W[1]-complete and `dominating set` is W[2]-complete, while `bandwidth` is W[t]-hard for all $t > 0$. If a parametric problem is W[t]-hard, this means that it is very unlikely that it is tractable. The higher the t for which W[t]-hardness is proved (or, at the limit, W[P]-hardness) the stronger the implication of intractability.

It should be noted that the W hierarchy, as defined in [5], does not appear to have the classification power of, say, NP-completeness theory and of the polynomial hierarchy, in that many natural problems are only partially classified, proved hard for one class and in another, higher one (or, as in the case of `bandwidth`, W[t]-hard for all $t > 0$ but not known to be in W[P]). This imperfect classification power is apparent in our results as well.

3 Parametric Complexity of Query Languages

We review briefly first basic definitions on databases and queries. A *database* $d = \{D; R_1, \ldots, R_m\}$ consists of a domain D and a set of relations $R_1, \ldots, R_m$ over D. A *query* Q is a function that maps a database d to a relation $Q(d)$ (of certain arity) over the same domain D. Queries are specified using *query languages*. A query language is capable of expressing a corresponding class of queries.

We will discuss in this paper the following languages (classes of queries): conjunctive queries, positive queries, first-order queries, and Datalog. Conjunctive queries correspond to relational algebra with selection, projection, join and renaming (or calculus with conjunction and existential quantification); positive queries add union (disjunction in calculus) to this list. First order queries add set difference (negation in calculus). Datalog adds recursion to the positive queries. We refer to the textbooks [13, 1] for a detailed exposition.

In the *evaluation problem* for a query Q, we are given database d and wish to compute $Q(d)$. In the *decision problem*, we are given in addition to the database d a tuple t, and wish to decide if $t \in Q(d)$. When discussing the complexity of these problems, we assume a standard encoding of databases and queries. The complexity of query languages is usually measured in database theory via the decision problem. The *combined complexity* of a query language Λ is the complexity of the decision problem (set) $\{(Q,d,t)|Q \in \Lambda, t \in Q(d)\}$. The *data complexity* of a query language Λ is the complexity of the sets $\{(d,t)|t \in Q(d)\}$, for queries $Q \in \Lambda$; that is, the query is regarded as fixed. Thus for example, the data complexity of a query language Λ is polynomial if there is a function $f : \Lambda \to N$ from queries to positive integers such that for every $Q \in \Lambda$, there is an algorithm which on input a database d of size n and a tuple t decides if $t \in Q(d)$ in time $O(n^{f(Q)})$.

In order to define the *parametric* complexity of query languages, we must first decide on the appropriate parameter to use. Two possible parameters come to mind: The *query size* q (the length of the string needed to express the query in Λ), and the *number of variables* v appearing in the query. In few cases it also matters if we assume that the *signature* (set of relations and their arity) is fixed or can vary. We will assume in the following by default a variable signature, and in the few cases where a fixed signature makes a difference we will mention what happens.

The relationship between these four parametric problems (the query complexity problem above parameterized with v as parameter, or with q as parameter, each with fixed or variable signature) is as depicted in the partial order below:

$$\begin{tikzcd}
 & v\text{, variable sig.} \arrow[dl, dash] \arrow[dr, dash] & \\
v\text{, fixed sig.} \arrow[dr, dash] & & q\text{, variable sig.} \arrow[dl, dash] \\
 & q\text{, fixed sig.} &
\end{tikzcd}$$

Figure 1

Proposition 1 *If one of the four parametric problems in Figure 1 is hard for a class in the W hierarchy, then all problems above it are also hard. If a problem is in some class in the W hierarchy, then all problems below it are also in the same class.*

Proof. The identity map is a valid parametric reduction for all four arcs in the partial order. □

4 A Classification of Query Languages

We consider the following four query languages: (1) Conjunctive queries; (2) Positive queries; (3) First-order queries; (4) Datalog queries. All these query languages are known to have query complexity AC_0 (the first three) and P (Datalog).

Theorem 1 *The parametric versions of the query evaluation problems correponding to these four query languages are classified as described in the table.*

query language	parameter: query size q	parameter: number of variables v
conjunctive	W[1]-complete	W[1]-hard, in W[2]
positive	W[1]-complete	W[SAT]-hard
first-order	W[t]-hard, all t	W[P]-hard
Datalog	W[P]-hard	W[P]-hard

Note: In the case of fixed signature, all the entries are the same except that the (conjunctive, parameter v) problem is in W[1] (and thus, W[1]-complete).

Sketch of proof. 1. *Conjunctive queries.* The lower bounds follow by a simple reduction from the `clique` problem, which is known to be W[1]-complete [5]. For any instance (G, k) of `clique` we construct a database consisting of one binary relation $G(.,.)$ (the graph). The query for parameter k is simply

$$P :- \bigwedge_{1 \le i < j \le k} G(x_i, x_j).$$

The goal proposition (0-ary relation) P is true iff G has a clique of size k. The query size is $q = O(k^2)$, while the number of variables is $v = k$, so this is a reduction to both problems. Note that this query is just a join of a set of binary relations.

For the upper bounds, in the case of parameter q, we can express any conjunctive query in 3-CNF by having Boolean variables that express the mapping from atoms of the query to tuples in the database. In the case of the parameter v, we have Boolean variables for the mapping from the query variables to the database constants. We omit the details from this abstract.

2. *Positive queries.* For the upper bound of W[1] (parameter q), we transform the query into a union of (exponentially many in q) conjunctive queries; note that in this case we need the full power of parametric reductions, as opposed to transformations. The W[SAT] lower bound (parameter v) is by a reduction from the `weighted formula satisfiability` problem (omitted).

3. *First-order queries.* The reduction is similar for the two cases. It is from the monotone `weighted circuit satisfiability` problem, which is W[P]-complete. We can assume that the given circuit alternates between OR and AND gates, and that the output is an OR gate,

at level $2t$. Our database contains only a binary relation C, describing the wiring diagram (dag) of the given circuit; the constants are gates (and therefore the variables will stand for gates). Define the following sequence of first-order queries, for the even (OR) levels of the circuit

$$\theta_0(x) = [C(x, x_1) \vee C(x, x_2) \vee \cdots \vee C(x, x_k)],$$

$$\theta_{2i}(x) = \exists y[C(x, y) \wedge \forall x(\neg C(y, x) \vee \theta_{2i-2}(x)].$$

Finally, the query is

$$Q = \exists x_1 \exists x_2 \ldots \exists x_k \theta_{2t}(o),$$

where o is the constant standing for the output gate Note: θ_{2t} is expanded fully using inductively the previous formulas in the sequence; the formula of the query has size $O(t)$ and uses $k + 2$ variables. Intuitively, $\theta_{2i}(x)$ means "OR gate x at level $2i$ is true, when inputs $x_1, x_2, \ldots, x_k$ are set to 1," and thus the query is true if and only if the given instance of `weighted circuit satisfiability` has a solution. Notice that a fixed signature (only a binary relation) is required.

4. *Datalog.* The reduction is from monotone `weighted circuit satisfiability` for both parameters. The database has a ternary relation $\mathrm{AND}(x, y, z)$ containing all triples of gates such that x is an AND gate with inputs y and z, and a binary relation $\mathrm{OR}(x, y)$ containing all pairs (x, y) such that x is an OR gate and y is one of its inputs. We also have a unary relation $\mathrm{I}(x)$, containing all input gates x. o is again the constant corresponding to the output gate. The query is the following:

$$\begin{array}{ll}
\mathrm{P}(x, v_1, \ldots, v_k) & \text{:- } \mathrm{P}(y, v_1, \ldots, v_k),\ \mathrm{P}(z, v_1, \ldots, v_k),\ \mathrm{AND}(x, y, z) \\
\mathrm{P}(x, v_1, \ldots, v_k) & \text{:- } \mathrm{P}(y, v_1, \ldots, v_k),\ \mathrm{OR}(x, y) \\
\mathrm{P}(v_1, v_1, v_2, \ldots, v_k) & \text{:- } \mathrm{I}(v_1), \ldots, \mathrm{I}(v_k) \\
\mathrm{P}(v_2, v_1, v_2 \ldots, v_k) & \text{:- } \mathrm{I}(v_1), \ldots, \mathrm{I}(v_k) \\
 & \vdots \\
\mathrm{P}(v_k, v_1, v_2 \ldots, v_k) & \text{:- } \mathrm{I}(v_1), \ldots, \mathrm{I}(v_k) \\
\mathrm{G} & \text{:- } \mathrm{P}(o, v_1, \ldots, v_k)
\end{array}$$

Obviously, the goal predicate G is true if and only if there is a setting of (at most) k input gates to 1 so that the output gate is 1. Note that the database (EDB) relations have all fixed arity, but the IDB relation P does not (it has arity $k + 1$). It can be shown that if we restrict all EDB and IDB relations to have fixed arity (independent of the parameter), then the problem is in W[1] (and thus W[1]-complete) for both parameters. □

5 A Tractable Case

Is there a nontrivial class of queries that is parametrically tractable? Even some simple queries that involve joins are NP-complete in combined complexity, and probably parametrically intractable as well, as we saw. Acyclic joins with projection and selection form the major exception to this. We will show in this section a nontrivial extension of that result.

Consider a conjunctive query Q:

$$G(t_0) : -R_{i_1}(t_1), \ldots, R_{i_s}(t_s)$$

Form a hypergraph H, which has the variables of Q as its nodes and has a hyperedge for every atom in the body of Q which contains the variables that occur in the atom. The query Q is called *acyclic* if the hypergraph H is acyclic. We can evaluate Q as follows. For every atom $R_{i_j}(t_j)$ in the body of Q, compute a relation S_j over the set of attributes corresponding to the variables of t_j such that a tuple is in S_j iff the corresponding instantiation of t_j is in relation R_{i_j} of the given database; S_j can be computed by performing appropriate selections and projection on R_{i_j}. Let Z be the set of attributes corresponding to the variables of the tuple t_0 in the head. Compute $\pi_Z(S_1 \bowtie \ldots \bowtie S_s)$ from which we can easily construct the result of the query $Q(d)$. If Q is acyclic, this evaluation can be done in time polynomial in the size of the input database d and the output $Q(d)$ [16]. If we only want to check whether $Q(d)$ is empty or whether a specific given tuple t is in $Q(d)$, we can do it in time polynomial in the size of d (substitute the constants of t in the body of the rule and evaluate the resulting query which will be either empty or contain one tuple, t).

Suppose now that in the body of the conjunctive query we have, in addition to the relational atoms, inequality atoms $x_i \neq x_j$ or $x_i \neq c$ between the variables or variables and constants. In this case we would normally include in the hypergraph also edges (x_i, x_j) corresponding to the inequalities between the variables (see [13]). However, inclusion of these edges destroys acyclicity even in very simple cases. Some examples: find the employees that work on more than one projects ($G(e) : -EP(e,p), EP(e,p'), p \neq p'$, where EP is the employee-project relation); Find the students that take courses outside their department ($G(s) : -SD(s,d)SC(s,c)CD(c,d')d \neq d'$). Of course, in general we may have more complicated queries with multiple relations and which may not be binary (i.e., a genuine hypergraph).

Suppose that we have a conjunctive query with inequalities and that the hypergraph defined by considering only the relational atoms is acyclic. We call this an acyclic query with inequalities. Is the combined complexity still polynomial? Unfortunately, not: the problem becomes NP-complete. For example, the Hamiltonian path problem can be easily reduced to it. Given a graph (V, E), let Q be the query

$$G : -E(x_1, x_2), E(x_2, x_3), \ldots, E(x_{n-1}, x_n), x_1 \neq x_2, x_1 \neq x_3, \ldots, x_{n-1} \neq x_n$$

The goal proposition (0-ary relation) G is true iff the graph is Hamiltonian. Here the query is as big as the database. However, in the more interesting case where the query is 'small', the problem remains tractable, but now in the fixed parameter (f.p.) sense.

Theorem 2 *The class of acyclic conjunctive queries with inequalities is f.p. tractable, both with respect to the query size and the number of variables as the parameter. Furthermore, we can evaluate such a query in f.p. polynomial time in the input and the output.*

A special case is the problem of finding simple paths of a specified length k in a graph. This problem was proved f.p. tractable by Monien [10], and an improved algorithm was given in [2] using an elegant "color-coding" (hashing) technique. Our algorithm combines this technique with acyclic query processing techniques.

The basic idea is to hash the domain D into a smaller domain (with size bounded by the number of variables), and use the hash values to check inequalities, while using the original values to check equality on the join attributes. Let Q be an acyclic query with inequalities, and let $H = (V, E)$ be its hypergraph. Partition the inequality atoms of Q into the set I_1 of atoms $x_i \neq x_j$ such that the variables x_i, x_j do not occur together in any hyperedge (relational atom), and the set I_2 of the remaining atoms ($x_i \neq c$ and $x_i \neq x_j$ such that x_i, x_j are in a common hyperedge). Let V_1 be the set of variables that occur in I_1 and let $k = |V_1|$. Let h be a function that maps D to the set $\{1, \ldots, k\}$. Consider an instantiation τ of the variables. We say that τ is *consistent* with h if for every inequality $x_i \neq x_j$ of I_1 we have $h(\tau(x_i)) \neq h(\tau(x_j))$; clearly this implies also that $\tau(x_i) \neq \tau(x_j)$, but not necessarily vice-versa. The instantiation τ is *satisfying* if it satisfies all the (relational and inequality) atoms in the body of Q. Let Θ_h be the set of all consistent satisfying instantiations, and let $Q_h(d) = \{\tau(t_0) | \tau \in \Theta_h\}$.

Fix a function $h : D \rightarrow \{1, \ldots, k\}$. We describe an f.p. polynomial time algorithm that decides whether there is a consistent satisfying instantiation τ and computes $Q_h(d)$. First, compute as above for each relational atom $R_{i_j}(t_j)$ of Q a corresponding relation, apply to it selections that incorporate the inequality atoms $x_i \neq c$ such that x_i occurs in t_j and $x_i \neq x_l$ such that both x_i, x_l occur in t_j, and let S_j be the resulting relation on set of attributes (variables) U_j. Let V_1' be a set of new attributes corresponding to V_1. If $X \subseteq V$ is a set of (original) variables, we use X' to denote the set of new attributes $\{x_i' | x_i \in X \cap V_1\}$. If t is a tuple over X, we can extend it to a tuple over XX' by letting $t[x_i'] = h(t[x_i])$ for each $x_i' \in X'$. Extend in this manner each relation S_j to a relation S_j' over the set of attributes $U_j U_j'$; note that S_j' has the same number of tuples as S_j and the new attributes take values in $\{1, \ldots, k\}$. For the emptiness problem, in essence what we will compute is the selection on inequalities of the projection on V_1' of the join of the relations S_j'. The selections and projections can be pushed inside the join for efficiency. In more detail we proceed as follows.

Let T be a join forest for H. Recall that this is a forest which has the hyperedges as its nodes, and with the property that for every attribute x_i, the set of nodes of T (i.e. hyperedges of H) that contain x_i form a connected subgraph (i.e. a subtree) T_i. We assume without loss of generality in the following that T is a tree (otherwise, for example, we can add a new dummy node corresponding to the empty hyperedge and connect it to a node in each component).

Root the tree at some node. For each node j of T, let W_j be the set of variables $x_i \in V_1 - U_j$ such that x_i appears in the subtree rooted at j - hence in a unique proper subtree rooted at a child of node j - and there is an inequality $x_i \neq x_l$ of I_1 such that x_l does not occur in the same proper subtree; in other words, node j separates the subtree T_i corresponding to x_i from the subtree T_l corresponding to x_l. Let $Y_j = U_j U_j' W_j'$. It is easy to see that the attribute sets Y_j form an acyclic hypergraph with the same tree T as its join tree.

To test if $Q_h(d) = \emptyset$, we perform a bottom-up pass of the tree as follows.

1. Initialize for each node $j \in T$ a relation $P_j := S'_j$.

2. Process all the nodes except the root in bottom-up order of T as follows. To process node j of T with parent u, compute $P_u := \sigma_F(P_u \bowtie \pi_{Y_j \cap Y_u}(P_j))$, where F is the conjunction of the inequalities $x'_i \neq x'_l$ such that $x'_i \in Y_j - U'_u$ and x'_l belongs to the attribute set of P_u at this point but not to Y_j. If $P_u = \emptyset$ then quit and report $Q_h(d) = \emptyset$.

3. If all nodes are processed successfully, then report $Q_h(d) \neq \emptyset$.

To compute $Q_h(d)$ (if it is not empty), we proceed as follows. At the end of the first pass we have a set of relations P_j over the attribute sets Y_j. It is not hard to see that the join of the P_j's is a relation over the attribute set VV'_1 that consists of all tuples $\tau\tau'_1$ such that τ is a satisfying instantiation that is consistent with h and τ'_1 is the extension of τ to V'_1. We do not actually want to compute the join (it is too large). We can reduce the relations P_j (and S_j, S'_j) by removing dangling tuples, i.e. tuples that do not participate in the join, using a downward pass. We process all the nodes except the root top-down. To process node j with parent u, set $P_j := P_j \ltimes P_u$.

We then perform a second bottom-up pass to compute $Q_h(d) = \pi_Z(P_1 \bowtie \ldots P_s)$, where Z is the set of variables that appear in the tuple t_0 of the head. In bottom-up order we process each nonroot node j, say with parent u, by setting $P_u := P_u \bowtie \pi_{Z_j}(P_j)$, where Z_j consists of $Y_j \cap Y_u$ and the attributes of Z that appear in the subtree rooted at node j. At the root r we compute $\pi_Z(P_r)$ which is $Q_h(d)$.

Consider a consistent instantiation τ and let l be the number of distinct values assumed by the variables. Then τ is consistent with at least a fraction $l!/l^k > e^{-k}$ of the functions h from D to $\{1, \ldots, k\}$. Thus, trying out a set of $O(e^k)$ random functions h will determine with high probability whether $Q(d) = \emptyset$. For a deterministic algorithm, we can use a k-perfect family F of hash functions, i.e., a family F which has the property that for every subset S of k (or less) elements of D, there is a $h \in F$ that hashes S into distinct values. One can construct such a family F with $2^{O(k)} \log |D|$ hash functions that can be evaluated in constant time (see [2] and the references therein). Then $Q(d) = \cup_{h \in F} Q_h(d)$. The time complexity of the algorithm for determining whether $Q(d) = \emptyset$ or whether a specific given tuple t is in $Q(d)$, is certainly bounded by $O(g(k) n \log^2 n)$, where $g(k) = 2^{O(k \log k)}$ and n is the size of the database; one $\log n$ factor is from sorting to perform the joins and the second from the perfect hash family. The time to compute $Q(d)$ is bounded by $O(g(k) nm \log^2 n)$ where $m = |Q(d)|$ is the size of the output.

If the parameter is q, the query size, the same theorem holds in the case where instead of a conjunction of inequalities in the body, we have a Boolean formula ϕ built from inequality atoms using $\vee$ and $\wedge$. If the parameter is v, the number of variables, then the problem becomes W[1]-hard if there are constants in ϕ, i.e., atoms $x_i \neq c$ combined arbitrarily, although it remains f.p. tractable if the atoms $x_i \neq c$ appear only conjunctively.

Can we extend the result to acyclic conjunctive queries with comparisons ($<$ or $\leq$) between variables or variables and constants? Example: Find the employees that have higher salary than their manager ($G(e) : -EM(e, m), ES(e, s), ES(m, s'), s' < s$). First, note that trivially any equality $x = y$ can be expressed as the conjunction of the two inequalities $x \leq y$ and $y \leq x$, so the question makes sense only if we first identify equal variables (otherwise, we can express trivially any conjunctive query by a set of atoms with disjoint variables and equalities). Given a conjunctive query Q with a set C of comparison atoms, we must first determine if C is consistent and find the implied equalities, between variables and constants which we then collapse. This is done (for dense orders) by forming a graph whose nodes are the variables and constants in C, with a directed arc $u \to w$ between two nodes u, w labeled $<$ or $\leq$ if C contains the corresponding constraint $u < w$ or $u \leq w$ or u, w are constants with $u < w$. The system is consistent iff there is no strongly connected component that contains a $<$ arc, and the implied equalities are that all nodes of the same strong component are equal (see eg. [8]). Let Q' be the resulting query after collapsing equal variables and constants of Q, and C' its set of comparison constraints (which is now acyclic). We say that the query is acyclic if the hypergraph corresponding to the relational atoms in the body of Q' is acyclic. Can we evaluate such a query in f.p. polynomial time? Unfortunately, not.

Theorem 3 *The class of acyclic conjunctive queries with comparisons is W[1]-hard with repsect to both parameters p and v.*

We omit the proof from the extended abstract. The theorem holds even in restricted cases (for binary relations, path queries, only $<$ constraints etc.)

References

[1] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] N. Alon, R. Yuster, U. Zwick, "Color-Coding", *J. ACM*, pp. 844-856, 1995.

[3] J. F. Buss, J. Goldsmith, "Nondeterminism within P", *SIAM J. Comput.*, pp. 560-572, 1993.

[4] A. K. Chandra, P. M. Merlin, "Optimal Implementation of Conjunctive Queries in Relational Databases", *Proc. 9th ACM Symp. Theory of Comp.*, pp. 77-90, 1977.

[5] R. G. Downey, M. R. Fellows, "Fixed-parameter Tractability and Completeness I: Basic Results", *SIAM J. Comp.*, pp. 873-921, 1995.

[6] P. C. Kanellakis, "Elements of Relational Database Theory", in *Handbook of Theoretical Computer Science*, J. Van Leeuwen ed., pp. 1074-1156, Elsevier, 1991.

[7] P. C. Kanellakis, "Constraint Programming and Database Languages: A Tutorial", *Proc. 14th ACM Symp. Principles of Database Sys.*, pp. 46-53, 1995.

[8] A. Klug, "On Conjunctive Queries Containing Inequalities", *J.ACM*, pp. 146-160, 1988.

[9] O. Lichtenstein, A. Pnueli, "Checking that Finite State Concurrent Programs Satisfy their Specifications", *Proc. 12th Annual ACM Symp. on Principles of Prog. Lang.*, pp. 97-107, 1985.

[10] B. Monien, "How to Find Long Paths Efficiently", *Ann. Disc. Math.*, pp. 239-254, 1985.

[11] C. H. Papadimitriou, M. Yannakakis, "On Limited Nondeterminism and the Complexity of the VC dimension", *J. Comp. Sys. Sc.*, pp. 161-170, 1996.

[12] N. Robertson, P. D. Seymour, "Graph Minors XIII: The Disjoint Paths Problem".

[13] J. D. Ullman, *Principles of Database and Knowledge Base Systems*, Computer Science Press, 1988.

[14] M. Y. Vardi, "The Complexity of Relational Query Languages", *Proc. ACM Symp. Theory of Computing*, pp. 137-146, 1982.

[15] M. Y. Vardi, "On the Complexity of Bounded-Variable Queries", *Proc. ACM Symp. Principles of Database Sys.*, pp. 266-276, 1995.

[16] M. Yannakakis, "Algorithms for Acyclic Database Schemes", *Proc. 7th Intl. Conf. Very Large Data Bases*, pp. 82-94, 1981.

[17] M. Yannakakis, "Perspectives on Database Theory", *Proc. IEEE Symp. Foundations of Comp. Sc.*, 1995.