Prediction-Hardness of Acyclic Conjunctive Queries [1]

Kouichi Hirata [2]

Department of Artificial Intelligence, Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan

Abstract

A *conjunctive query problem* is a problem to determine whether or not a tuple belongs to the answer of a *conjunctive query* over a database. In this paper, a tuple, a conjunctive query and a database in relational database theory are regarded as a ground atom, a nonrecursive function-free definite clause and a finite set of ground atoms, respectively, in inductive logic programming terminology. An *acyclic conjunctive query problem* is a conjunctive query problem with *acyclicity*. Concerned with the acyclic conjunctive query problem, in this paper, we present the hardness results of predicting acyclic conjunctive queries from an *instance* with a j*-database* of which predicate symbol is at most j-ary. Also we deal with two kinds of instances, a *simple instance* as a set of ground atoms and an *extended instance* as a set of pairs of a ground atom and a description. We mainly show that, from both a simple and an extended instances, acyclic conjunctive queries are not polynomial-time predictable with j-databases ($j \geq 3$) under the cryptographic assumptions, and predicting acyclic conjunctive queries with 2-databases is as hard as predicting DNF formulas. Hence, the acyclic conjunctive queries become a natural example that the equivalence between subsumption-efficiency and efficient pac-learnability from both a simple and an extended instances collapses.

Key words: acyclic conjunctive query, inductive logic programming, prediction, prediction-preserving reduction, subsumption

[1] A preliminary version of the paper appeared in the *Proceedings of the 11th International Conference on Algorithmic Learning Theory*, LNAI **1968** (Springer, 2000) 238–251.

[2] Partially supported by Grand-in-Aid for Scientific Research 15700137 and 16016275 from the Ministry of Education, Culture, Sports, Science and Technology, Japan, and 13558036 from the Japan Society for the Promotion of Science.

1 Introduction

From the viewpoints of both learning theory and inductive logic programming, Džeroski *et al.* [11] have first shown the learnability of (first-order) definite programs called *ij-determinate*. Furthermore, the series of the researches by Cohen [5–7,9], Džeroski [10,12,23], Kietz [22–24] and Page [9,28] have placed the theoretical researches for the learnability of logic programs in one of the main research topics in inductive logic programming. Recently, they have deeply developed by many researchers [1,20,25,26,31,32].

On the other hand, a *conjunctive query problem* in relational database theory [2,4,14,17,36] is a problem to determine whether or not a *tuple* belongs to the answer of a *conjunctive query* over a *database*. Here, a tuple, a conjunctive query, and a database in relational database theory are regarded as a ground atom $e = p(t_1, \ldots, t_n)$, a nonrecursive function-free definite clause $C = p(x_1, \ldots, x_n) \leftarrow A_1, \ldots, A_m$, and a finite set B of ground atoms in inductive logic programming terminology. Then, we can say that the conjunctive query problem is a problem to determine whether or not e is provable from C over B, i.e., $\{C\} \cup B \vdash e$.

Since database schemes in relational database theory can be viewed as *hypergraphs*, many researchers such as [2,4,13,14,17,36] have widely investigated the properties of database schemes and hypergraphs, together with the *acyclicity* of them. A hypergraph is called *acyclic* if it reduces to an empty hypergraph by *GYO*-reduction (See Section 2 bellow[3]).

It is known that the acyclicity frequently makes intractable problems in cyclic cases tractable. The conjunctive query problem is such an example. Here, a conjunctive query is called *acyclic* if its associated hypergraph is acyclic. While the conjunctive query problem is NP-complete in general [15], Yannakakis has shown that it becomes solvable in polynomial time if a conjunctive query is acyclic [36]. Recently, Gottlob *et al.* have improved the Yannakakis's result that it is LOGCFL-complete [17].

Concerned with the conjunctive query problem, in this paper, we present the hardness results of predicting *acyclic conjunctive queries* from an *instance* with a *database*. In particular, we deal with a *j-database* of which predicate symbol is at most j-ary.

According to Cohen [5–7], we introduce two kinds of instances, a *simple* and an *extended instances*. A simple instance of (C, B), which comes from a general setting in learning theory, is a set $\{e \mid \{C\} \cup B \vdash e\}$. On the other hand, an extended instance of (C, B), which comes from a proper setting in inductive

[3] Note here that the concept of acyclicity is different from one in [1,31].

logic programming, is a set $\{(e, D) \mid \{C\} \cup B \cup D \vdash e\}$, where D is a set of ground unit clauses and called a *description*. If an extended instance is allowed, then many programs that are usually written with function symbols can be rewritten as function-free programs. Furthermore, some experimental learning systems such as Foil [30] also impose a similar restriction.

The acyclic conjunctive query problem, which is LOGCFL-complete mentioned above, is corresponding to the *evaluation* problem of our prediction problem. Schapire [34] has shown that, if the corresponding evaluation problem is NP-hard, then the prediction problem is not polynomial-time predictable unless NP $\subseteq$ P/Poly. Hence, we cannot apply Schapire's result to show the hardness results in our problem. In order to achieve the prediction-hardness, we adopt the *prediction-preserving reduction* [29] as similar as Cohen [6,7].

As the prediction-hardness from a *simple instance*, we show that, for each $j \geq 3$, acyclic conjunctive queries are not polynomial-time predictable with j-databases under the cryptographic assumptions that inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are solvable in polynomial time. Also, as same as Cohen's proof (Lemma 11 in [7] or Theorem 4 in [9]), we point out that predicting acyclic conjunctive queries with 2-databases is as hard as predicting DNF formulas. The latter hardness result holds even if their depth is at most 1. Furthermore, we show that acyclic conjunctive queries are polynomial-time pac-learnable with 1-databases.

As the prediction-hardness from an *extended instance*, we give the similar results as above. We show that, for each $j \geq 3$, acyclic conjunctive queries are not polynomial-time predictable with j-databases under the cryptographic assumptions, and predicting acyclic conjunctive queries (of which depth is at most 2) with 2-databases is as hard as predicting DNF formulas.

Our hardness results imply that acyclic conjunctive queries become a natural example that the equivalence between subsumption-efficiency and efficient pac-learnability from both a simple and an extended instances collapses. In general, the subsumption problem for conjunctive queries is NP-complete [3,15]. It is also known that, for both famous *determinate* [11] and *k-local* [7,9] conjunctive queries, the subsumption problems are solvable in polynomial time [24]. As the learnability results, k-local conjunctive queries are polynomial-time pac-learnable from a simple instance, while determinate conjunctive queries are not polynomial-time predictable from a simple instance under the cryptographic assumptions [7]. Note that the determinate conjunctive queries are defined over *ordered* conjunctive queries, so it is slightly artificial. On the other hand, for acyclic conjunctive queries, while the subsumption problem is LOGCFL-complete [17], it is not polynomial-time predictable from both a simple and an extended instances under the cryptographic assumptions.

2 Preliminaries

In this paper, a *term* is either a constant symbol or a variable. An *atom* is of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and each t_i is a term. A *literal* is an atom or the negation of an atom. A *positive literal* is an atom and a *negative literal* is the negation of an atom. A *clause* is a disjunction of literals. A *definite clause* is a clause containing one positive literal. A *unit clause* is a clause consisting of just one positive literal. By the definition of a term, a clause is always *function-free*.

A definite clause C is represented as

$$A \leftarrow A_1, \ldots, A_m \quad \text{or} \quad A \leftarrow A_1 \wedge \ldots \wedge A_m,$$

where A and A_i $(1 \leq i \leq m)$ are atoms. Here, an atom A is called the *head* of C and denoted by $hd(C)$, and a set $\{A_1, \ldots, A_m\}$ is called the *body* of C and denoted by $bd(C)$.

A definite clause C is *ground* if C contains no variables. A definite clause C is *nonrecursive* if each predicate symbol in $bd(C)$ is different from one of $hd(C)$, and *recursive* otherwise[4] . Furthermore, a finite set of ground unit clauses is called a *database*. A database is called a *j-database* if each predicate symbol in it is at most j-ary. According to the convention of relational database theory [2,14,17,36], in this paper, we call a nonrecursive definite clause containing no constant symbols a *conjunctive query*.

A *substitution* is a partial function mapping variables to constant symbols or variables. We will represent substitutions with the Greek letters θ and σ and (when necessary) write them as sets $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ where x_i is a variable and t_i is a term $(1 \leq i \leq n)$. For a literal A, $A\theta$ denotes the result of replacing each variable x_i in A with t_i. If θ and σ are substitutions, we will use $A\theta\sigma$ to denote $(A\theta)\sigma$.

Next, we formulate the concept of acyclicity. A hypergraph $H = (V, E)$ consists of a set V of vertices and a set $E \subseteq 2^V$ of hyperedges. For a hypergraph $H = (V, E)$, the *GYO-reduct* $GYO(H)$ [2,13,14,17] of H is the hypergraph obtained from H by repeatedly applying the following rules as long as possible:

(1) Remove hyperedges that are empty or contained in other hyperedges.
(2) Remove vertices that appear in ≤ 1 hyperedges.

Definition 1 (*cf.* [2,13,14,17]) A hypergraph H is called *acyclic* if $GYO(H)$

[4] A recursive clause in this paper is sometimes called an *ambivalent* clause [16].

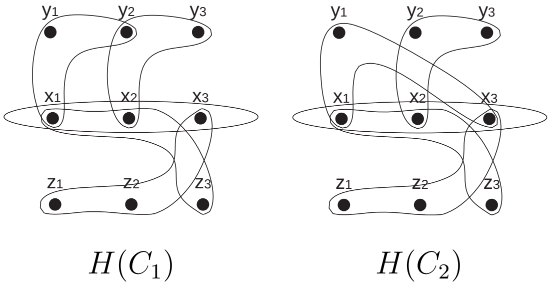

Fig. 1. The associated hypergraphs $H(C_1)$ and $H(C_2)$ to C_1 and C_2.

is an empty hypergraph, i.e., $GYO(H) = (\emptyset, \emptyset)$, and *cyclic* otherwise.

The *associated hypergraph* $H(C)$ *to* a conjunctive query C is a hypergraph

$$(var(C), \{var(L) \mid L \in bd(C)\}),$$

where $var(S)$ denotes the set of all variables occurring in S [5].

Definition 2 (Gottlob *et al.* [17]) A conjunctive query C is called *acyclic* (*resp.*, *cyclic*) if the associated hypergraph $H(C)$ to C is acyclic (*resp.*, cyclic).

Example 3 *Let C_1 and C_2 be the following conjunctive queries.*

$$\begin{aligned} C_1 &= p(x_1, x_2, x_3) \\ &\qquad \leftarrow q(x_1, y_1, \underline{y_2}), r(x_2, y_2, y_3), q(x_3, z_1, z_2), r(x_1, x_2, z_3), s(x_1, x_2, x_3), \\ C_2 &= p(x_1, x_2, x_3) \\ &\qquad \leftarrow q(x_1, y_1, \underline{x_3}), r(x_2, y_2, y_3), q(x_3, z_1, z_2), r(x_1, x_2, z_3), s(x_1, x_2, x_3). \end{aligned}$$

Then, the associated hypergraphs $H(C_1)$ and $H(C_2)$ to C_1 and C_2 are described as Figure 1. By the GYO-reduction, we can show that

$$GYO(H(C_1)) = (\{x_1, x_2, \underline{y_2}\}, \{\{x_1, x_2\}, \{x_1, \underline{y_2}\}, \{x_2, \underline{y_2}\}\}) \neq (\emptyset, \emptyset),$$

but $GYO(H(C_2)) = (\emptyset, \emptyset)$, so C_1 is cyclic but C_2 is acyclic.

Gottlob *et al.* [17] have shown that the problem of determining whether or not a conjunctive query or a hypergraph is acyclic is in symmetric logspace SL.

Let C be a conjunctive query $A \leftarrow A_1, \ldots, A_m$, B a database and e a ground atom. Then, we say that e is *provable* from C over B and denote it by $\{C\} \cup B \vdash$

[5] In the preliminary version [19], the associated hypergraph is given as $(var(C), \{var(L) \mid L \in C\})$, which is different from the definition in [17] represented here. Hence, the results in Section 4 are different from ones in [19], whereas the results in Section 5 hold under both definitions.

e if one of the following conditions holds.

(1) $e \in B$.
(2) There exists a substitution θ such that $e = A\theta$ and $\{A_1\theta, \ldots, A_m\theta\} \subseteq B$.

Consider the following decision problem [6]:

ACQ (Acyclic Conjunctive Query) [17]
INSTANCE: An acyclic conjunctive query $C = p(x_1, \ldots, x_n) \leftarrow A_1, \ldots, A_m$, a database B, and a ground atom $e = p(t_1, \ldots, t_n)$.
QUESTION: Does $\{C\} \cup B \vdash e$ hold?

Theorem 4 (Gottlob *et al.* **[17])** *The problem* **ACQ** *is* LOGCFL*-complete.*

The relationship between LOGCFL and other relevant complexity classes is summarized in the following chain of inclusions:

$$\mathrm{AC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{DL} \subseteq \mathrm{SL} \subseteq \mathrm{NL} \subseteq \mathrm{LOGCFL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2 \subseteq \mathrm{NC} \subseteq \mathrm{P} \subseteq \mathrm{NP}.$$

In the remainder of this section, we introduce some classes of conjunctive queries.

Let C be a conjunctive query. The *free variables* of C are variables in $bd(C)$ but not in $hd(C)$. A conjunctive query C is *k-free* [7] if C has at most k free variables.

Let x and y be free variables of C. x is *adjacent to* y if they appear in the same literal of C. x is *connected to* y if either x is adjacent to y or there exists a variable z such that x is adjacent to z and z is connected to y. The *locale* of a variable x is the set of literals that contain either x or some variable adjacent to x. The *locality* of C is the cardinality of the largest locale of any free variable in C. A conjunctive query C is *k-local* [7,9] if the locality of C is at most k.

A conjunctive query $C = A \leftarrow A_1, \cdots, A_m$ is called *ordered* if the order from 1 to m in C is fixed. Let C be an ordered conjunctive query $A \leftarrow A_1, \ldots, A_m$ and B be a database. Then, C is *determinate w.r.t.* B [11] if for each i $(1 \leq i \leq n)$ and substitution θ such that

$$A\theta \leftarrow A_1\theta, \ldots, A_{i-1}\theta \text{ is ground and } B \models A_1\theta \wedge \ldots \wedge A_{i-1}\theta,$$

there exists *at most one* substitution σ such that

$$A_i\theta\sigma \text{ is ground and } B \models A_i\theta\sigma.$$

[6] Gottlob *et al.* [17] have called the problem **ACQ** "Acyclic Conjunctive Query Output Tuple (ACQOT)".

The *depth* of a variable x in a conjunctive query $C = A \leftarrow A_1, \ldots, A_m$ is defined as follows: If x occurs in A, then the depth of x in C is 0. Suppose that x first occurs in A_i. If none of the other variables in A_i already occurred in $A \leftarrow A_1, \ldots, A_{i-1}$, then the depth of x in C is ∞. Otherwise, the depth of x in C is 1 plus the depth of the variable in A_i with greatest depth occurring in $A \leftarrow A_1, \ldots, A_{i-1}$. The *depth* of C is the largest depth of the variable in C. A conjunctive query C is *k-depth* [11] if the depth of C is at most k.

3 Models of Learnability

In this section, we introduce the models of learnability. The definitions and notations in this section are mainly due to Cohen [6,7].

Let C be a conjunctive query and B be a database. A ground atom e is a *fact* of C if the predicate symbol of e is same as one of $hd(C)$. In this paper, assume that there exists no element of B of which predicate symbol is same as $hd(C)$.

For a conjunctive query C and a database B, the following set is called a *simple instance* of (C, B):

$$\{e \mid \{C\} \cup B \vdash e, e \text{ is a fact of } C\}.$$

For an element e of a simple instance of (C, B), we say that e is *covered by* (C, B).

Furthermore, we introduce a *description* D, which is a finite set of ground unit clauses. Then, the following set of pairs is called an *extended instance* of (C, B):

$$\{(e, D) \mid \{C\} \cup D \cup B \vdash e, e \text{ is a fact of } C\}.$$

For an element (e, D) of an extended instance of (C, B), we say that (e, D) is *covered by* (C, B).

In his learnability results, Cohen has adopted both the simple instance [7] and the extended instance [5,6]. If the extended instance is allowed, then many programs that are usually written with function symbols can be rewritten as function-free programs. There is also a close relationship between extended instances and "flattening" [10,18,26,33]. Some experimental learning systems such as FOIL [30] also impose a similar restriction. See the papers [5,6] for more detail.

In the following, we introduce some definitions and notions in learning theory.

Let X be a set, called a domain. Define a concept c over X to be a representation of some subset of X, and a language L to be a set of concepts. Associated with X and L are two *size complexity measures*. We will write the size complexity of some concept $c \in L$ or instance $e \in X$ as $|c|$ or $|e|$, and we will assume that this complexity measure is polynomially related to the number of bits needed to represent c or e. We use the notation X_n (*resp.*, L_n) to stand for the set of all elements of X (*resp.*, L) of size complexity no greater than n.

An *example* of c is a pair (e, b), where $b = 1$ if $e \in c$ and $b = 0$ otherwise. If d is a probability distribution function, a *sample of c from X drawn according to d* is a pair of multisets S^+, S^- drawn from the domain X according to d, S^+ containing only positive examples of c, and S^- containing only negative examples of c.

Definition 5 (Cohen [6,7]) A language L is *polynomial-time predictable* if there exists an algorithm PacPredict and a polynomial $m(1/\varepsilon, 1/\delta, n_e, n_t)$ so that for every $n_t > 0$, every $n_e > 0$, every $c \in L_{n_t}$, every ε $(0 \leq \varepsilon \leq 1)$, every δ $(0 \leq \delta \leq 1)$, and every probability distribution function d, PacPredict has the following behavior:

(1) Given a sample S^+, S^- of c from X_{n_e} drawn according to d and containing at least $m(1/\varepsilon, 1/\delta, n_e, n_t)$ examples, PacPredict outputs a hypothesis h such that
$$prob(d(h - c) + d(c - h) > \varepsilon) < \delta,$$
where the probability is taken over the possible samples S^+ and S^-.
(2) PacPredict runs in time polynomial in $1/\varepsilon$, $1/\delta$, n_e, n_t, and the number of examples.
(3) h can be evaluated in polynomial time.

The algorithm PacPredict is called a *prediction algorithm* for L.

Definition 6 (Cohen [6,7]) A language L is *polynomial-time pac-learnable* if there exists an algorithm PacLearn so that:

(1) PacLearn satisfies all the requirements in Definition 5, and
(2) on inputs S^+ and S^-, PacLearn always outputs a hypothesis $h \in L$.

We will abbreviate "polynomial-time predictable" and "polynomial-time pac-learnable" as "predictable" and "pac-learnable," respectively.

For a language L, it is known that, if L is pac-learnable, then L is predictable, but the converse does not hold in general. Hence, if L is not predictable, then L is not pac-learnable.

In this paper, a language L is regarded as some set of conjunctive queries.

Furthermore, for a database B, $L[B]$ denotes the set of pairs of the form (C, B) such that $C \in L$. Semantically, such a pair denotes either a simple or an extended instance covered by it. Furthermore, we will deal with the following languages:

(1) ACQ denotes the set of all *acyclic* conjunctive queries.
(2) k-FreeCQ denotes the set of all *k-free* conjunctive queries.
(3) k-$\mathsf{LocalCQ}$ denotes the set of all *k-local* conjunctive queries.
(4) DetCQ denotes the set of all *determinate* conjunctive queries.
(5) k-$\mathsf{DepthCQ}$ denotes the set of all *k-depth* conjunctive queries.

For some set B of databases, $L[\mathsf{B}]$ denotes the set $\{L[B] \mid B \in \mathsf{B}\}$. Such a set of languages is called a *language family*. Also the set of j-databases is denoted by j-B.

Definition 7 (Cohen [6,7]) A language family $L[\mathsf{B}]$ is *predictable* if for every $B \in \mathsf{B}$ there exists a prediction algorithm $\textsc{PacPredict}_B$ for $L[B]$. The pac-learnability of a language family is defined similarly.

Schapire [34] has shown that, if the evaluation problem is NP-hard, then the prediction problem is not predictable unless NP $\subseteq$ P/Poly. Since the problem **ACQ** is corresponding to an evaluation problem for the prediction problem of $\mathsf{ACQ}[\mathsf{B}]$ and it is LOGCFL-complete, we cannot apply Schapire's result to our prediction problem.

Pitt and Warmuth [29] have introduced a notion of reducibility between prediction problems. *Prediction-preserving reducibility* is essentially a method of showing that one language is no harder to predict than another.

Definition 8 (Pitt & Warmuth [29]) Let L_i be a language over domain X_i $(i = 1, 2)$. We say that *predicting L_1 reduces to predicting L_2*, denoted by $L_1 \trianglelefteq L_2$, if there exists a function $f : X_1 \rightarrow X_2$ (called an *instance mapping*) and a function $g : L_1 \rightarrow L_2$ (called a *concept mapping*) satisfying the following conditions:

(1) $x \in c$ iff $f(x) \in g(c)$.
(2) The size complexity of g is polynomial in the size complexity of c.
(3) $f(x)$ can be computed in polynomial time.

Theorem 9 (Pitt & Warmuth [29]) *Suppose that $L_1 \trianglelefteq L_2$.*

(1) If L_2 is predictable, then so is L_1.
(2) If L_1 is not predictable, then neither is L_2.

For some polynomial p, let $\mu\mathsf{BF}_n^{p(n)}$ be the class of read-once Boolean formulas, that is, Boolean formulas in which each variable occurs at most once, over n

Boolean variables of size at most $p(n)$. Let $\mu\mathsf{BF}^{p(n)} = \bigcup_{n\geq 1} \mu\mathsf{BF}_n^{p(n)}$. Then:

Theorem 10 (Kearns & Valiant [21]) $\mu\mathsf{BF}^{p(n)}$ *is not predictable under the cryptographic assumptions that inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are solvable in polynomial time.*

Let DNF_n be the class of DNF formulas over n Boolean variables, and let $\mathsf{DNF} = \bigcup_{n\geq 1} \mathsf{DNF}_n$. It remains open whether or not DNF is predictable.

Finally, we summarize the previous results for the learnability of restricted conjunctive queries from a simple instance.

Theorem 11 *The following statements hold:*

(1) **(Cohen [7], Cohen & Page [9])** k-$\mathsf{FreeCQ}[j\text{-}\mathsf{B}]$ $(k \geq 1, j \geq 2)$ *is predictable from a simple instance iff* DNF *is predictable.*
(2) **(Cohen [7], Cohen & Page [9])** k-$\mathsf{LocalCQ}[j\text{-}\mathsf{B}]$ $(k \geq 0, j \geq 0)$ *is pac-learnable from a simple instance.*
(3) **(Cohen [7], Cohen & Page [9])** k-$\mathsf{DepthCQ}[j\text{-}\mathsf{B}]$ $(k \geq 1, j \geq 2)$ *is not predictable from a simple instance unless* $\mathrm{NP} \subseteq \mathrm{P/Poly}$.
(4) **(Cohen [7])** $\mathsf{DetCQ}[j\text{-}\mathsf{B}]$ $(j \geq 3)$ *is not predictable from a simple instance under the cryptographic assumptions.*
(5) **(Džeroski *et al.* [11])** k-$\mathsf{DepthDetCQ}[j\text{-}\mathsf{B}]$ $(k \geq 0, j \geq 0)$ *is pac-learnable from a simple instance.*

4 Prediction-Hardness of Acyclic Conjunctive Queries from a Simple Instance

In this section, we discuss the prediction-hardness of acyclic conjunctive queries from a *simple instance*.

As the related previous results, if we can receive a fact as a *ground clause*, Kietz [22,23] implicitly has shown that acyclic conjunctive queries consisting of literals with at most j-ary predicate symbols $(j \geq 2)$ are not pac-learnable unless RP = PSPACE, without databases as background knowledge. Under the same setting, Cohen [8] has strengthened this result not to be predictable under the cryptographic assumptions.

First, we obtain the following theorem. Note that the following proof is motivated by Cohen (Theorem 5 in [6] and Theorem 9 in [7]).

Theorem 12 $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ $(j \geq 3)$ *is not predictable from a simple instance under the cryptographic assumptions.*

PROOF. By Theorem 9 and 10, it is sufficient to show that, for each $n \geq 0$, there exists a database $B \in \textsf{3-B}$ such that $\mu\textsf{BF}_n^{p(n)} \trianglelefteq \textsf{ACQ}[B]$ from a simple instance.

Let $e = e_1 \dots e_n \in \{0,1\}^n$ be a truth assignment and $F \in \mu\textsf{BF}_n^{p(n)}$ be a Boolean formula of size polynomial $p(n)$ over Boolean variables $\{x_1, \dots, x_n\}$. First, construct the following database $B \in \textsf{3-B}$:

$$B = \left\{ \begin{array}{l} \mathit{and}(0,0,0),\ \mathit{and}(0,1,0),\ \mathit{or}(0,0,0),\ \mathit{or}(0,1,1),\ \mathit{not}(0,1), \\ \mathit{and}(1,0,0),\ \mathit{and}(1,1,1),\ \mathit{or}(1,0,1),\ \mathit{or}(1,1,1),\ \mathit{not}(1,0) \end{array} \right\}.$$

Also construct the following instance mapping f:

$$f(e) = q(e_1, \dots, e_n, 1).$$

Note that F is represented as a tree of size polynomial $p(n)$ such that each internal node is labeled by $\wedge$, $\vee$ or $\neg$, and each leaf is labeled by a Boolean variable in $\{x_1, \dots, x_n\}$. Each internal node n_i of F ($1 \leq i \leq p(n)$) has one (n_i is labeled by $\neg$) or two (n_i is labeled by $\wedge$ or $\vee$) input variables and one output variable y_i. Let L_i be the following literals:

$$L_i = \begin{cases} \mathit{and}(z_{i1}, z_{i2}, y_i) & \text{if } n_i \text{ is labeled by } \wedge, \\ \mathit{or}(z_{i1}, z_{i2}, y_i) & \text{if } n_i \text{ is labeled by } \vee, \\ \mathit{not}(z_{i1}, y_i) & \text{if } n_i \text{ is labeled by } \neg. \end{cases}$$

Here, z_{i1} and z_{i2} denote input variables of n_i. Then, construct the following concept mapping g:

$$g(F) = q(x_1, \dots, x_n, y) \leftarrow (\textstyle\bigwedge_{1 \leq i \leq p(n)} L_i),$$

where y is a variable in $(\bigwedge_{1 \leq i \leq p(n)} L_i)$ corresponding to an output of F.

Since F is represented as a tree, the associated hypergraph $H(g(F))$ of $g(F)$ is acyclic, so $g(F)$ is acyclic. Furthermore, it holds that e satisfies F iff $f(e)$ is covered by $(g(F), B)$, that is,

$$\{g(F)\} \cup B \vdash f(e).$$

Hence, it holds that $\mu\textsf{BF}_n^{p(n)} \trianglelefteq \textsf{ACQ}[B]$ from a simple instance. □

For $\textsf{ACQ}[\textsf{2-B}]$, we obtain the following weaker hardness result than Theorem 12.

Theorem 13 (Cohen [7], Cohen & Page [9]) *If* $\mathsf{ACQ}[2\text{-}\mathsf{B}]$ *is predictable from a simple instance, then so is* DNF.

Theorem 13 follows from the proof of only-if direction of the statement 1 in Theorem 11, that is, for each $n \geq 1$, there exists a 2-database $B \in 2\text{-}\mathsf{B}$ such that $\mathsf{DNF}_n \trianglelefteq 1\text{-}\mathsf{FreeCQ}[B]$ (Lemma 12 in [7] or Theorem 4 in [9]), because the 1-free conjunctive query in this reduction is acyclic. An extension of this proof will be presented as the proof of Theorem 17.

Note that the conjunctive query in this reduction is also 1-depth. Then, as the prediction-hardness of depth-bounded acyclic conjunctive queries from a simple instance, the following corollary holds.

Corollary 14 *If* k-$\mathsf{DepthACQ}[j\text{-}\mathsf{B}]$ $(k \geq 1, j \geq 2)$ *is predictable from a simple instance, then so is* DNF.

By Theorem 13 and Corollary 14, we can conclude that predicting $\mathsf{ACQ}[2\text{-}\mathsf{B}]$ and k-$\mathsf{DepthACQ}[j\text{-}\mathsf{B}]$ $(k \geq 1, j \geq 2)$ from a simple instance is as hard as predicting DNF.

For $\mathsf{ACQ}[1\text{-}\mathsf{B}]$, we also obtain the following theorem.

Theorem 15 $\mathsf{ACQ}[1\text{-}\mathsf{B}]$ *is pac-learnable from a simple instance.*

PROOF. We can assume that a target acyclic conjunctive query has no variables that occur in the body but not in the head. Let n be an arity of a target predicate q, and m be the number of distinct predicate symbols in $B \in 1\text{-}\mathsf{B}$, where m predicate symbols are denoted by $q_1, \ldots, q_m$. We set an initial hypothesis C as:

$$C = q(x_1, \ldots, x_n) \leftarrow \textstyle\bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq m} q_j(x_i).$$

Then, by applying Valiant's technique of learning monomials [35] to C, the statement holds. □

5 Prediction-Hardness of Acyclic Conjunctive Queries from an Extended Instance

In this section, we discuss the prediction-hardness of acyclic conjunctive queries from an *extended instance*.

By using Cohen's result (Theorem 3 in [6]), we can claim that, for each $j \geq 3$, the recursive version of $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ is not predictable from an *extended* *instance* under the cryptographic assumptions. In contrast, the following theorem claims that this statement also holds for the nonrecursive version of $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ ($j \geq 3$).

Theorem 16 $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ ($j \geq 3$) *is not predictable from an extended instance under the cryptographic assumptions.*

PROOF. By Theorem 9 and 10, it is sufficient to show that, for each $n \geq 0$, there exists a database $B \in 3\text{-}\mathsf{B}$ such that $\mu\mathsf{BF}_n^{p(n)} \trianglelefteq \mathsf{ACQ}[B]$ from an extended instance.

First, we give e, F and B as same as the proof of Theorem 12. By the definition of an extended instance, an instance mapping f must map e to a pair of a fact and a description. Then, construct the following instance mapping f:

$$f(e) = (q(1), \{bit_1(e_1), \dots, bit_n(e_n)\}).$$

By using the same literals L_i ($1 \leq i \leq p(n)$) as Theorem 12, construct the following concept mapping g:

$$g(F) = q(y) \leftarrow (\textstyle\bigwedge_{1 \leq j \leq n} bit_j(x_j)), (\textstyle\bigwedge_{1 \leq i \leq p(n)} L_i).$$

Here, y is a variable in $(\bigwedge_{1 \leq i \leq p(n)} L_i)$ corresponding to an output of F.

Since F is represented as a tree, the associated hypergraph $H(g(F))$ of $g(F)$ is acyclic, so $g(F)$ is acyclic. Furthermore, it holds that e satisfies F iff $f(e)$ is covered by $(g(F), B)$, that is,

$$\{g(F)\} \cup \{bit_1(e_1), \dots, bit_n(e_n)\} \cup B \vdash q(1).$$

Hence, it holds that $\mu\mathsf{BF}_n^{p(n)} \trianglelefteq \mathsf{ACQ}[B]$ from an extended instance. □

For $\mathsf{ACQ}[2\text{-}\mathsf{B}]$, as similar as Theorem 13, we also obtain the following weaker hardness result than Theorem 16.

Theorem 17 *If* $\mathsf{ACQ}[2\text{-}\mathsf{B}]$ *is predictable from an extended instance, then so is* DNF.

PROOF. It is sufficient to show that, for each $n \geq 1$, there exists a 2-database $B \in 2\text{-}\mathsf{B}$ such that $\mathsf{DNF}_n \trianglelefteq \mathsf{ACQ}[B]$ from an extended instance. This proof is an extension of the proof of Lemma 12 in [7] or Theorem 4 in [9].

Let $e = e_1 \dots e_n \in \{0,1\}^n$ be a truth assignment and $F \in \mathsf{DNF}_n$ be a DNF formula $(l_1^1 \wedge \dots \wedge l_{m_1}^1) \vee \dots \vee (l_1^k \wedge \dots \wedge l_{m_k}^k)$ with k terms over Boolean variables

$\{x_1, \ldots, x_n\}$, where l^i_j denotes a literal, that is, either a Boolean variable or a negation of a Boolean variable.

First, construct the following database $B \in$ 2-B. Here, $\mathbf{k}$ denotes the set $\{1, \ldots, k\}$.

$$\begin{aligned} B = & \bigcup_{1\leq i\leq k} \{\mathit{true}_i(1,l) \mid l \in \mathbf{k}\} \cup \bigcup_{1\leq i\leq k} \{\mathit{true}_i(0,l) \mid l \in \mathbf{k} - \{i\}\} \\ & \cup \bigcup_{1\leq i\leq k} \{\mathit{false}_i(0,l) \mid l \in \mathbf{k}\} \cup \bigcup_{1\leq i\leq k} \{\mathit{false}_i(1,l) \mid l \in \mathbf{k} - \{i\}\} \\ & \cup \{\mathit{lit}(l,1) \mid l \in \mathbf{k}\}. \end{aligned}$$

Note that the size of B is polynomially bounded by the size of F.

Construct the same instance mapping f as the proof of Theorem 16, that is,

$$f(e) = (q(1), \{\mathit{bit}_1(e_1), \ldots, \mathit{bit}_n(e_n)\}).$$

Furthermore, construct the following concept mapping g:

$$g(F) = q(y) \leftarrow (\textstyle\bigwedge_{1\leq h\leq n} \mathit{bit}_h(x_h)), (\bigwedge_{1\leq i\leq k}(\bigwedge_{1\leq j\leq m_i} M^i_j)), \mathit{lit}(z,y),$$

where M^i_j $(1 \leq i \leq k, 1 \leq j \leq m_i)$ is defined as follows:

$$M^i_j = \begin{cases} \mathit{true}_i(x^i_j, z) & \text{if } l^i_j = x^i_j, \\ \mathit{false}_i(x^i_j, z) & \text{if } l^i_j = \overline{x^i_j}. \end{cases}$$

It is obvious that $g(F)$ is acyclic and the size of $g(F)$ is polynomially bounded by the size of F. Note that, for each $l \in \mathbf{k}$, the l-th term of F is satisfied by the truth assignment $e_1 \ldots e_n$ iff the variable z can be substituted to l when x_h is substituted to e_h $(1 \leq h \leq n)$. Then, it holds that e satisfies F iff $f(e)$ is covered by $(g(F), B)$, that is,

$$\{g(F)\} \cup \{\mathit{bit}_1(e_1), \ldots, \mathit{bit}_n(e_n)\} \cup B \vdash q(1).$$

Hence, it holds that $\mathsf{DNF}_n \trianglelefteq \mathsf{ACQ}[B]$ from an extended instance. □

Note that we can array the atoms in $\mathit{bd}(g(F))$ as the 2-depth acyclic conjunctive query. Then, as the prediction-hardness of depth-bounded acyclic conjunctive queries from an extended instance, the following corollary holds.

Corollary 18 *If k-*$\mathsf{DepthACQ}[j\text{-}\mathsf{B}]$ $(k \geq 2, j \geq 2)$ *is predictable from an extended instance, then so is* DNF.

By Theorem 17 and Corollary 18, we can conclude that predicting $\mathsf{ACQ}[2\text{-}\mathsf{B}]$ and k-$\mathsf{DepthACQ}[j\text{-}\mathsf{B}]$ ($k \geq 1, j \geq 2$) from an extended instance is as hard as predicting DNF.

6 Subsumption-Efficiency and Efficient Learnability

We say that a clause C_1 *subsumes* another clause C_2 if there exists a substitution θ such that $C_1\theta \subseteq C_2$. The *subsumption problem for a language* L is the problem of whether or not C_1 subsumes C_2 for each $C_1, C_2 \in L$.

The subsumption problem for conjunctive queries is NP-complete in general [3,15]. As the tractable cases for the subsumption problem, it is known the following theorem.

Theorem 19 *The following statements hold:*

(1) **(Kietz & Lübbe [24])** *The subsumption problems for* DetCQ *and* k-$\mathsf{LocalCQ}$ ($k \geq 0$) *are solvable in polynomial time.*
(2) **(Gottlob** *et al.* **[17])** *The subsumption problem for* ACQ *is* LOGCFL-*complete.*

By incorporating Theorem 19 with the statement 4 in Theorem 11, the language DetCQ is an example that the equivalence between subsumption-efficiency and efficient pac-learnability from a simple instance collapses. Note that DetCQ is defined over *ordered* conjunctive queries, which is slightly artificial; Consider the following database $B = \{m(a,c), m(b,c), f(c,e), f(d,e)\}$. Then, $gf(x,z) \leftarrow f(y,z), m(x,y)$ is determinate w.r.t. B, while $gf(x,z) \leftarrow m(x,y), f(y,z)$ is not [23]. Hence, there exist two same clauses without their order such that one is determinate but another is not.

On the other hand, by incorporating Theorem 19 with Theorem 12 and 16, the language ACQ is an example that the equivalence between subsumption-efficiency and efficient pac-learnability from both a simple instance and an extended instance collapses. Since conjunctive queries are assumed not to be ordered for the ACQ, it is more natural than the DetCQ.

Note that we cannot directly extend the statement 4 in Theorem 11 to the prediction-hardness from an extended instance under the *determinacy with respect to databases*, although we can extend Theorem 12 and 13 to Theorem 16 and 17, respectively. If we adopt the *determinacy with respect to databases and descriptions*, that is, if we can regard descriptions as background knowledge, then determinate conjunctive queries are not predictable from an extended instance (*cf.* the proof of Theorem 9 in [7]).

7 Conclusion

In this paper, we have mainly discussed the hardness results of predicting acyclic conjunctive queries from both a simple and an extended instances.

As the prediction-hardness from a *simple instance*, we have shown that $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ $(j \geq 3)$ is not polynomial-time predictable under the cryptographic assumptions. Also, as same as Cohen's proof (Lemma 11 in [7] or Theorem 4 in [9]), we have pointed out that predicting $\mathsf{ACQ}[2\text{-}\mathsf{B}]$ and k-$\mathsf{DepthACQ}[j\text{-}\mathsf{B}]$ $(k \geq 1, j \geq 2)$ is as hard as predicting DNF. Furthermore, we have shown that $\mathsf{ACQ}[1\text{-}\mathsf{B}]$ is polynomial-time pac-learnable from a *simple instance*.

As the prediction-hardness from an *extended instance*, we have given the similar hardness results. We have shown that $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ $(j \geq 3)$ is not polynomial-time predictable under the cryptographic assumptions, and predicting $\mathsf{ACQ}[2\text{-}\mathsf{B}]$ and k-$\mathsf{DepthACQ}[j\text{-}\mathsf{B}]$ $(k \geq 2, j \geq 2)$ is as hard as predicting DNF.

The above prediction-hardness implies that the language ACQ becomes a natural example that the equivalence between subsumption-efficiency and efficient pac-learnability from both a simple and an extended instances collapses.

Various researches have investigated the efficient learnability by using equivalence and membership queries such as [1,25,26,32,31]. Our result in this paper implies that $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ $(j \geq 3)$ is not polynomial-time learnable using equivalence queries alone. It is a future work to analyze the learnability of $\mathsf{ACQ}[j\text{-}\mathsf{B}]$ $(j \geq 3)$ by using membership and equivalence queries, and by extending to one containing function symbols or recursion. It is also a future work to analyze the relationship between our acyclicity and the acyclicity introduced by [1,31].

Fagin [14] has given the *degree* of acyclicity; α-acyclic, β-acyclic, γ-acyclic and Berge-acyclic. In particular, he has shown the following chain of implication for any hypergraph H: H is Berge-acyclic $\Rightarrow$ H is γ-acyclic $\Rightarrow$ H is β-acyclic $\Rightarrow$ H is α-acyclic (none of the reverse implication holds in general). Acyclicity in the literature such as [2,4,13,17,36] and also in this paper is corresponding to Fagin's α-acyclicity [14]. Note that none of the results in this paper implies the predictability of the other degrees of acyclicity, while all of the corresponding evaluation problems are LOGCFL-complete [17]. It is a future work to investigate the relationship between the degree of acyclicity and the learnability.

Acknowledgment

The author would like to thank Hiroki Arimura in Hokkaido University for a motivation of this paper and insightful comments. He also would like to thank Akihiro Yamamoto in Kyoto University and Shinichi Shimozono in Kyushu Institute of Technology for constructive discussion. Finally, he would like to thank anonymous referees of ALT2000 for valuable comments to revise the preliminary version [19] of this paper.

References

[1] H. Arimura, Learning acyclic first-order Horn sentences from entailment, in: *Proceedings of the 8th International Workshop on Algorithmic Learning Theory*, LNAI **1316** (Springer, 1997) 432–445.

[2] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, *J. ACM* **30**(3) (1983) 479–513.

[3] L. D. Baxter, *The complexity of unification*, Doctoral Thesis, Department of Computer Science, University of Waterloo, 1977.

[4] C. Chekuri, A. Rajaraman, Conjunctive query containment revisited, *Theor. Comput. Sci.* **239** (2000) 211–229.

[5] W. W. Cohen, Pac-learning recursive logic programs: Efficient algorithms, *J. Artif. Intell. Res.* **2** (1995) 501–539.

[6] W. W. Cohen, Pac-learning recursive logic programs: Negative results, *J. Artif. Intell. Res.* **2** (1995) 541–573.

[7] W. W. Cohen, Pac-learning non-recursive Prolog clauses, *Artif. Intell.* **79**(1) (1995) 1–38.

[8] W. W. Cohen, The dual DFA learning problem: Hardness results for programming by demonstration and learning first-order representations, in: *Proceedings of the 9th Annual Workshop on Computational Learning Theory* (ACM, 1996) 29–40.

[9] W. W. Cohen, C. D. Page Jr., Polynomial learnability and inductive logic programming: Methods and results, *New Gener. Comput.* **13**(3-4) (1995) 369–409.

[10] L. De Raedt, S. Džeroski, First-order *jk*-clausal theories are PAC-learnable, *Artif. Intell.* **70**(1-2) (1994) 375–392.

[11] S. Džeroski, S. Muggleton, S. Russell, PAC-learnability of determinate logic programs, in: *Proceedings of the 5th Annual Workshop on Computational Learning Theory* (ACM, 1992) 128–135.

[12] S. Džeroski, S. Muggleton, S. Russell, Learnability of constrained logic programs, in: *Proceedings of the 6th European Conference on Machine Learning*, LNAI **667** (Springer, 1993) 342–347.

[13] T. Eiter, G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, *SIAM J. Comput.* **24**(6) (1995) 1278–1304.

[14] R. Fagin, Degrees of acyclicity for hypergraphs and relational database schemes, *J. ACM* **30**(3) (1983) 514–550.

[15] M. R. Garey, D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness* (W. H. Freeman and Company, 1979).

[16] G. Gottlob, Subsumption and implication, *Inform. Proc. Let.* **24**(2) (1987) 109–111.

[17] G. Gottlob, N. Leone, F. Scarcello, The complexity of acyclic conjunctive queries, *J. ACM* **43**(3) (2001) 431–498.

[18] K. Hirata, Flattening and implication, in: *Proceedings of the 10th International Conference on Algorithmic Learning Theory*, LNAI **1720** (Springer, 1999) 157–168.

[19] K. Hirata, On the hardness of learning acyclic conjunctive queries, in: *Proceedings of the 11th International Conference on Algorithmic Learning Theory*, LNAI **1968** (Springer, 2000) 238–251.

[20] T. Horváth, G. Turán, Learning logic programs with structured background knowledge, in: L. De Raedt (ed.), *Advances in inductive logic programming*, (IOS Press, 1996) 172–191.

[21] M. Kearns, L. Valiant, Cryptographic limitations on learning Boolean formulae and finite automata, *J. ACM* **41**(1) (1994) 67–95.

[22] J.-U. Kietz, Some lower bounds for the computational complexity of inductive logic programming, in: *Proceedings of the 6th European Conference on Machine Learning*, LNAI **667** (Springer, 1993) 115–123.

[23] J.-U. Kietz, S. Džeroski, Inductive logic programming and learnability, *SIGART Bull.* **5** (1994) 22–32.

[24] J. U. Kietz, M. Lübbe, An efficient subsumption algorithm for inductive logic programming, in: *Proceedings of the 11th International Conference on Machine Learning* (Morgan Kaufmann, 1994) 130–138.

[25] R. Khardon, Learning function-free Horn expressions, *Mach. Learn.* **35**(1) (1999) 241–275.

[26] R. Khardon, Learning range-restricted Horn expressions, in: *Proceedings of the 4th European Conference on Computational Learning Theory*, LNAI **1572** (Springer, 1999) 111–125.

[27] S. Muggleton (*ed.*), *Inductive logic programming* (Academic Press, 1992).

[28] C. D. Page Jr., A. M. Frisch, Generalization and learnability: A study of constrained atoms, in: [27] l29–161.

[29] L. Pitt, M. K. Warmuth, Prediction-preserving reduction, *J. Comp. System Sci.* **41**(3) (1990) 430–467.

[30] J. R. Quinlan, Learning logical definitions from relations, *Mach. Learn.* **5**(3) (1990) 239–266.

[31] C. Reddy, P. Tadepalli, Learning first-order acyclic Horn programs from entailment, in: *Proceedings of the 8th International Conference on Inductive Logic Programming*, LNAI **1446** (Springer, 1998) 23–37.

[32] C. Reddy, P. Tadepalli, Learning Horn definitions: Theory and application to planning, *New Gener. Comput.* **17**(1) (1999) 77–98.

[33] C. Rouveirol, Extensions of inversion of resolution applied to theory completion, in: [27] 63–92.

[34] R. E. Schapire, The strength of weak learning, *Mach. Learn.* **5**(2) (1990) 197–227.

[35] L. Valiant, A theory of learnable, *Comm. ACM* **27**(11) (1984) 1134–1142.

[36] M. Yannakakis, Algorithms for acyclic database schemes, in: *Proceedings of the 7th International Conference on Very Large Data Bases* (IEEE Computer Society, 1981) 82–94.