Fitting Algorithms for Conjunctive Queries

Balder ten Cate
ILLC, University of Amsterdam

Maurice Funk
Leipzig University and ScaDS.AI

Jean Christoph Jung
TU Dortmund University

Carsten Lutz
Leipzig University and ScaDS.AI

ABSTRACT

A fitting algorithm for conjunctive queries (CQs) is an algorithm that takes as input a collection of data examples and outputs a CQ that fits the examples. In this column, we propose a set of desirable properties of such algorithms and use this as a guide for surveying results from the authors' recent papers published in PODS 2023, IJCAI 2023, and *Inf. Proc. Letters* 2024. In particular, we explain and compare several concrete fitting algorithms, and we discuss complexity and size bounds for constructing fitting CQs with desirable properties.

1. INTRODUCTION

The *fitting problem* for conjunctive queries (CQs) is the problem to construct a CQ q that fits a given set of labeled data examples. This fundamental problem has a long history in database research. It lies at the heart of the classic *Query-By-Example* paradigm [41] that aims to assist users in query formation and query refinement, and that is also known as *query reverse engineering*. It has been intensively studied for CQs [38, 33, 7] and other types of queries (e.g., [10, 4, 20]). The fitting problem is also central to *Inductive Logic Programming* [21, 27], where CQs correspond to the basic case of non-recursive single-rule Datalog programs, and has close connections to fitting problems for *schema mappings* [2, 12]. More recent motivation comes from *automatic feature generation in machine learning with relational data* [30, 8]. Here, the CQ fitting problem arises because a CQ that separates positive from negative examples in (a sufficiently large subset of) a labeled dataset is a natural contender for being added as an input feature to the model [8].

Examples illustrating the fitting problem are given in Table 1. In each example, certain values from the database instance are labeled as positive and/or negative, and a fitting query must include the positive examples in its output and exclude the negative examples.

Depending on the application, desirable properties of a fitting algorithm may include the following:

Efficiency Ideally, the fitting algorithm should run in polynomial time in the size of the input examples, or at least in polynomial time in the size of the input examples plus the size of the smallest fitting query. We call the latter *weakly polynomial*.

Succinctness Ideally, the fitting algorithm outputs a query of small size, i.e., not much larger than the smallest fitting query. This can be formalized through the notion of the "Occam property", which requires that the size of the output concept is bounded polynomially in the size of a target query and sublinearly in the number of input examples.

Producing Extremal Fittings When several fitting queries exist, it may be desirable to output a query that is either most-general or most-specific among all fitting queries.

Generalization to Unseen Examples Ideally, we would like the fitting algorithm to come with a (probabilistic) guarantee that the output query not only fits the input examples, but performs well on future unseen examples (drawn from the same distribution as the input examples and labeled according to the same "target query"). This is formalized by the well-known PAC ("Probably Approximately Correct") property.

Completeness for Design Ideally, the fitting algorithm can be compelled to produce any CQ (up to equivalence) by giving it the right input examples.

It turns out that some of these properties are difficult or impossible to attain. Also, there are properties that can be obtained in isolation, but not in combination. A fundamental difficulty is that the fitting problem is computationally hard, and it is known that the smallest fitting CQ for a given collection of labeled examples is in general exponential in size [40]. To make things worse, the hardness pertains already to very small subclasses of CQs and natural restrictions on the fitting problem obtained by requiring, for instance, that all input examples use the same database instance (cf. Section 4).

[1]This paper was written in response to a dual invitation from the SIGMOD Record Database Principles column and the ACM SIGMOD Research Highlight Award committee.

Example database instance I:

Businessman
donald
fred
james

Democrat
barack
franklin

Father	
barack-sr	barack
fred	donald
james	franklin

Economist
barack-sr

Republican
donald

Labeled examples:	A fitting query:
$E^+ = \{(I, \text{franklin}), (I, \text{barack})\}$ $E^- = \{(I, \text{donald})\}$	$q(x)$:- Democrat(x).
$E^+ = \{(I, \text{franklin}), (I, \text{donald})\}$ $E^- = \{(I, \text{barack})\}$	$q(x)$:- Father(y, x), Businessman(y).
$E^+ = \{(I, \text{barack}), (I, \text{donald})\}$ $E^- = \{(I, \text{franklin})\}$	the union of CQs $q(x)$:- Republican(x) $q(x)$:- Father(y, x), Economist(y).

Table 1: Examples of the fitting problem

$$\begin{tikzcd}
\text{Strongly Occam} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Occam} \arrow[r, Rightarrow] & \text{Polynomial PAC} \arrow[dl, dash, "\times"] \arrow[d, dash, "\times"] \\
\text{Complete for Design} & \text{Producing a most-specific fitting} \arrow[l, Rightarrow] \arrow[r, dash, "\times"] & \text{Producing a most-general fitting when it exists}
\end{tikzcd}$$

Figure 1: Summary of interactions between desirable properties (the edge label $\times$ means incompatibility)

Despite these roadblocks, there are ways forward. One may opt to bite the bullet and live with the high computational complexity, for example by employing SAT solvers. One may also extend the capabilities of the fitting algorithm by allowing it to interactively ask questions to an oracle that provides black-box access to a fitting CQ (the "target CQ"). Specifically, the algorithm may produce a database instance and a tuple and ask whether the tuple belongs to the output of the target CQ on that instance. In reality, the oracle may be a preexisting compiled CQ (in reverse-engineering scenarios) or a user (in interactive query specification settings). A third option is to resort to incomplete approaches that do not always return a fitting CQ when one exists.

In this column, we (i) compare several different fitting algorithms based on the above considerations and (ii) discuss structural interactions between the above desiderata. In particular, we will study three fitting algorithms for CQs, described at a high level as follows:

Algorithm P This algorithm takes as input a collection of labeled examples and outputs the canonical CQ of the direct product of the positive examples, which is known to fit whenever a fitting CQ exists.

Algorithm M can be viewed as an optimized version of Algorithm P that additionally interacts with an oracle as described above. It computes the product of the positive examples in an iterative way and uses the oracle to minimize the query in each iteration.

Algorithm B proceeds by solving a size-bounded version of the fitting problem for increasing size bounds, until a fitting CQ is found. This is also known as the "bounded fitting approach".

Each of the five desirable properties listed above will be discussed in a separate section of this column. We will discuss how the three algorithms fare, explain how this is complemented by lower bounds, and discuss structural interactions between the different desiderata. Table 2 provides a summary of the interactions between the considered properties. The technical results we present are mostly drawn from the recent papers [14, 17, 16]. Indeed, our motivation for writing this column is to fit these results into a unified picture.

Fitting problems for CQs and especially also for broader classes of logic programs have been investigated extensively in the literature on *Inductive Logic Programming (ILP)*, with a strong emphasis on systems implementation. The most common technique used in ILP systems is based on *refinement operators*. In the case of CQs it is known that fitting algorithms based on refinement operators are incomplete. Nevertheless, refinement-operator based approaches have been successfully implemented and used. Therefore, in Section 9, we also discuss them and their relationship to our algorithms.

Outline. In Section 2, we define the fitting problem. In Section 3 we describe the three algorithms in more detail. In Sections 4–8, we investigate the above five desiderata one by one. In Section 9, we discuss refinement-based approaches to fitting. Finally, we conclude in Section 10.

2. THE FITTING PROBLEM FOR CQS

As usual, a schema $\mathcal{S}$ is a set of relation symbols, each with associated arity. A *database instance* over $\mathcal{S}$ is a finite set I of *facts* of the form $R(a_1, \ldots, a_n)$ where $R \in \mathcal{S}$ is a relation symbol of arity n and $a_1, \ldots, a_n$ are *values*. We use $adom(I)$ to denote the set of all values used in I. We can then view a *query* over a schema $\mathcal{S}$, semantically, as a function q that maps each database instance I over $\mathcal{S}$ to a set of k-tuples $q(I) \subseteq adom(I)^k$, where $k \geq 0$ is the *arity* of the query.

Data Examples. A *data example* for a query q consists of a database instance I together with information about the intended query output $q(I)$. We focus on two types of data examples (cf. Remark 2.4 for other types):

- a *positive example* for q is a pair $(I, \mathbf{a})$ with $\mathbf{a} \in q(I)$;
- a *negative example* for q is a pair $(I, \mathbf{a})$ with $\mathbf{a} \in adom(I)^k \setminus q(I)$

	Needs oracle	Running time	Extremal fitting	Occam	PAC	Complete for design
Alg P	no	Exponential in the input (*not* weakly polynomial)	yes (most-specific)	no	no	yes
Alg M	yes	Exponential in the input (weakly polynomial)	no	yes (strongly)	yes (polynomially)	yes
Alg B	no	Doubly exponential (*not* weakly polynomial)	no	yes (strongly)	yes (polynomially)	yes

Table 2: Summary of our comparison of fitting algorithms

with k the arity of the query q. We also say that k is the arity of the example.

By a *collection of labeled examples* we mean a pair $E = (E^+, E^-)$, where E^+ and E^- are sets of examples. We say that a query q *fits* E if each member of E^+ is a positive example for q and each member of E^- is a negative example for q, or, in other words, if $\mathbf{a} \in q(I)$ for all $(I, \mathbf{a}) \in E^+$ and $\mathbf{a} \notin q(I)$ for all $(I, \mathbf{a}) \in E^-$. Note that for a fitting to exist, all examples in the collection must have the same arity.

EXAMPLE 2.1. *Consider the database instance I depicted in Table 1, over a schema that consists of unary and binary relation symbols, storing information about American presidents. Table 1 lists several collections of labeled examples, and, for each, a fitting query. Note that in the third case, no fitting conjunctive query exists, but a fitting union of conjunctive queries exists.*

Fitting Problems. We discuss three algorithmic problems related to fitting, relative to a query language $\mathcal{Q}$:

Fitting Construction Given a collection E of labeled examples that has a fitting query in $\mathcal{Q}$, construct such a query.

Fitting Existence Given a collection E of labeled examples, decide whether a fitting query from $\mathcal{Q}$ exists.

Fitting Verification Given a collection E of labeled examples and a query $q \in \mathcal{Q}$, decide if q fits E.

Among the three problems above, fitting construction is the main problem of interest and we sometimes refer to it also as the *fitting problem.* Note that we have formulated fitting construction as a promise problem, in order to study its complexity in isolation from the fitting existence problem. In practice, one may combine a fitting construction algorithm with a fitting existence test.

A solution to the fitting construction problem, as defined above, is *any* query that fits. In particular, the query is not required to generalize from the input collection of labeled examples to other examples, i.e., there is no penalty for overfitting.

If $\mathcal{Q}$ is the class of all relational algebra queries, the above problems are trivial. Indeed, if constants are admitted in the query, a fitting relational algebra query exists for (E^+, E^-) if and only if $E^+ \cap E^- = \emptyset$; as a fitting query one can pick, intuitively, the union of the complete descriptions of the positive examples. Without constants, a fitting relational algebra query exists if and only if no member of E^+ is isomorphic to a member of E^- (cf. [23]). This situation changes for more restricted query languages $\mathcal{Q}$; we consider conjunctive queries.

Conjunctive Queries and tree CQs. By a k-ary *conjunctive query (CQ)* over a schema $\mathcal{S}$, we mean an expression of the form $q(\mathbf{x})$:- $\alpha_1, \ldots, \alpha_n$ where $\mathbf{x} = x_1, \ldots, x_k$ is a sequence of variables and each α_i is a relational atom that uses a relation symbol from $\mathcal{S}$ and no constants. The variables in $\mathbf{x}$ are called *answer variables* and the other variables used in the atoms α_i are the *existential variables.* Each answer variable is required to occur in at least one atom α_i, a requirement known as the *safety condition.* A CQ of arity 0 is called *Boolean.* With the *size* of a CQ, we mean the number of atoms in it. The query output $q(I)$ is defined as usual, cf. any standard database textbook. Two CQs q_1 and q_2 are *equivalent*, written $q_1 \equiv q_2$ if $q_1(I) = q_2(I)$ for all database instances I.

Besides CQs we will also consider a more restricted class of queries, namely *tree CQs.* In order to define it, we need to talk about canonical database instances. The canonical database instance of a CQ q is the instance I_q (over the same schema as q) whose active domain consists of the variables that occur in q and whose facts are the atomic formulas in q. The definition of tree CQs is restricted to schemas that consist of unary and binary relation symbols only. Note that every instance over such a schema can naturally be viewed as a directed, edge-labeled and node-labeled graph. A *tree CQ* is then a unary CQ $q(x)$ such that I_q, viewed in the above way, is a directed node-labeled and edge-labeled tree with root x (without parallel edges, and where all edges are directed away from the root). Our interest in this class of CQs stems from the fact that they form a notational variant of concept expressions in the description logic $\mathcal{EL}$, and that they are in some ways computationally more attractive.

The fitting construction, fitting existence, and fitting verification problems have been studied extensively for the case of CQs as well as for the case of tree CQs. We will review the known results in the subsequent sections. Here, we briefly comment on the fitting verification problem. It was observed in [14] that this problem is DP-complete for CQs, where DP is the class of problems that are the intersection of a problem in NP and a problem in coNP. Tree CQs, on the other hand, can be evaluated in polynomial time (combined complexity) and it follows that the fitting verification problem for tree CQs is solvable in polynomial time (cf. [14]). In summary:

THEOREM 2.2 ([14]). *Fitting verification is DP-complete for CQs and in PTime for tree CQs.*

REMARK 2.3. *As defined above, CQs do not contain constants, so we may not write, for instance, $q(x)$:- $R(x, 100)$. Filipetto [22] studied the impact of allowing constants on the fitting problem. For CQs, it turns out, whether we allow constants does not affect the fitting problem, modulo simple reductions. Specifically, given a collection of labeled examples E, we can take an isomorphic copy E' of E in which every constant is renamed, and take the union $E \cup E'$. Then (assuming E contains at least one positive example) the CQs that fit $E \cup E'$ are precisely the CQs that do not contain constants and fit E. Conversely, constants can be simulated by unary relations. The same applies to tree CQs.*

While we do not consider unions of conjunctive queries (UCQs) here, it is interesting to point out that they behave differently: allowing constants in UCQs trivializes the fitting problem. There are several ways to address this, including specifying a set of allowed constants as part of the input to the fitting problem, or restricting the number of allowed constants in the query. Both variants can be solved by a reduction to the constant-free fitting problem. Furthermore, identifying a small set of meaningful constants from data is an interesting problem by itself that deserves further study. See [22] for more details.

REMARK 2.4. *Depending on the application scenario, it may be natural to consider other types of examples besides positive and negative examples. In particular, an* input-output example *is a pair $(I, q(I))$ consisting of a database instance together with the entire query output. Such an example can be viewed as a succinct representation of a collection of $|adom(I)|^k$ many positive and negative examples, and therefore all the fitting algorithms that we will discuss can also be applied to input-output examples, although the complexity bounds do not necessarily carry over. See also [13, Section 6].*

Also, depending on the application scenario, it may be the case that E^+ and E^- consist of examples with the same database instance. Many of the complexity bounds we will discuss hold already in this restricted setting.

3. THREE FITTING ALGORITHMS

We now define in detail the three algorithms for the fitting construction problem for CQs.

Algorithm 3.1: Algorithm P

```
Input  : collection $(E^+, E^-)$ of labeled examples for
         which a fitting CQ exists
Output : a fitting CQ
1 $e^* := e^k_\top$ where $k$ is the arity of $(E^+, E^-)$;
2 foreach $e \in E^+$ do
3     $e^* := e^* \times e$
4 return the canonical CQ of $e^*$
```

Algorithm P. This algorithm simply returns the canonical CQ of the direct product of the positive examples. Intuitively, it extracts the commonalities of the positive examples. We make this more precise.

The *canonical CQ* of a data example $(I, a_1, \ldots, a_n)$ is simply the CQ $q(x_{a_1}, \ldots, x_{a_n})$ whose atoms are the facts of I, where each value $a \in adom(I)$ is uniformly replaced by a fresh variable x_a.

The *direct product* $I \times J$ of two instances I and J (over the same schema), is the database instance over $\mathcal{S}$ that consists of all facts $R(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_n, b_n\rangle)$, where $R(a_1, \ldots, a_n)$ is a fact in I and $R(b_1, \ldots, b_n)$ is a fact in J. Note that the active domain of $I \times J$ consists of pairs from $adom(I) \times adom(J)$. The direct product $(I, \mathbf{a}) \times (J, \mathbf{b})$ of two examples, with $\mathbf{a} = a_1, \ldots, a_k$ and $\mathbf{b} = b_1, \ldots, b_k$ of the same length, is given by $(I \times J, (\langle a_1, b_1\rangle, \langle a_2, b_2\rangle \ldots, \langle a_k, b_k\rangle)$. In general, this may not yield a well-defined example because there is no guarantee that the distinguished elements $\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_k, b_k\rangle$ belong to $adom(I \times J)$. For a finite set of examples $E = \{e_1, \ldots, e_n\}$, we write $\prod_{e \in E}(e)$ for the direct product $e_1 \times \cdots \times e_n$ (note that $\times$ is associative up to isomorphism). The direct product operation should not be confused with the Cartesian product operation from relational algebra: the former preserves the schema, including the arity of each relation, but changes the domain; the latter preserves the domain but produces a relation of increased arity.

We need one more notion. With $e^k_\top$, we denote the strongest example of arity k. More precisely, $e^k_\top$ takes the form $(I, \mathbf{a})$ where I is an instance with a single value a that contains all possible facts over the schema and a, and $\mathbf{a}$ is the tuple $(a, \ldots, a)$ of length k. With these notions in place, Algorithm P is given as Algorithm 3.1. It simply computes $\Pi_{e \in E^+}(e)$, the negative examples are not used. Note that taking the product of n instances may result in an active domain of size exponential in n.

EXAMPLE 3.1. *In almost all of the remaining examples in this paper and without further notice, we use a schema $\mathcal{S}$ that consists of a single binary relation symbol R and consider Boolean queries. Note that, then, we may view an example simply as a database instance, without distinguished values. Also, many examples will refer to database instances that take the form of a cycle. For every $i \geq 1$, we use C_i to denote the database instance that*

Algorithm 3.2: Algorithm M

```
Input  : collection $(E^+, E^-)$ of labeled examples for
         which a fitting CQ exists
Output : a fitting CQ
1 $e^* := e_\top^k$ where $k$ is the arity of $(E^+, E^-)$;
2 foreach $e \in E^+$ do
3     $e^* := e^* \times e$;
4     $e^* :=$ minimize$(e^*)$;
5 return the canonical CQ of $e^*$
6 procedure minimize$(I, \mathbf{a})$
7 foreach $f \in I$ do
8     if $\mathbf{a} \in \widehat{q}(I \setminus \{f\})$ according to $MEMB_{\widehat{q}}$ then
9         $I := I \setminus \{f\}$
```

consists of the facts $R(a_1, a_2), R(a_2, a_3), \ldots, R(a_i, a_1)$. In other words, C_i is a cycle with i edges.

Now consider the collection of labeled examples $E = (E^+, E^-)$ where $E^+ = \{C_2, C_3\}$ and E^- consists of the single-fact instance $\{R(a, b)\}$. Then Algorithm P outputs the canonical CQ of C_6. Note that the direct product of any two instances C_n and C_m with n, m prime is isomorphic to $C_{n \cdot m}$. Clearly, this is a fitting CQ for E.

The following well-known fact (cf. for instance [40]) implies the correctness of Algorithm P.

THEOREM 3.2. *If any CQ fits a collection of labeled examples $E = (E^+, E^-)$, then the canonical CQ of the direct product $\Pi_{e \in E^+}(e)$ is well-defined and fits E.*

This also implies that Algorithm P can be turned into an algorithm for fitting existence by checking whether the constructed CQ fits the negative examples.

Algorithm M. We next consider Algorithm M, first given in [17]. Algorithm M differs from Algorithms P and B in having access via a membership oracle (whence the "M") to a concrete target CQ $\widehat{q}$ that fits the collection of examples given as an input. Given an example e, the oracle returns (in unit time) the status of e, that is, whether e is a positive or negative example for $\widehat{q}$. We denote such an oracle with $MEMB_{\widehat{q}}$. Membership oracles play a central role in exact learning in the style of Angluin [3]. Note that, just like the other two algorithms, Algorithm M only needs to solve the fitting construction problem: it may return *any* CQ that fits the input examples, not necessarily one that is equivalent to $\widehat{q}$.

Algorithm M is given as Algorithm 3.2. It works essentially in the same way as Algorithm P except that, between any two product constructions, it minimizes the constructed example. With the latter, we mean to drop facts as long as the oracle $\text{MEMB}_{\widehat{q}}$ tells us that the resulting example is still positive for $\widehat{q}$. This relies on the invariant that, during the run of the **foreach** loop in Line 2, all constructed examples e^* are positive for $\widehat{q}$.

It is not hard to see that Algorithm M returns a CQ q such that $q \subseteq \widehat{q}$, i.e., $q(I) \subseteq \widehat{q}(I)$ for all instances I.

Algorithm 3.3: Algorithm B

```
Input  : collection $(E^+, E^-)$ of labeled examples for
         which a fitting CQ exists
Output : a fitting CQ
1 foreach $s = 1, 2, \ldots$ do
2     if there is a fitting CQ of size $s$ then
3         return a fitting CQ of size $s$
```

However, q need not be equivalent to $\widehat{q}$. If q_Π is the CQ returned on the same input by Algorithm P, then $q_\Pi \subseteq q$, but again the two CQs need not be equivalent.

EXAMPLE 3.3. *Consider again the collection of data examples $E = (E^+, E^-)$ from Example 3.1, that is, $E^+ = \{C_2, C_3\}$ and E^- contains the single instance $\{R(a, b)\}$. The output of Algorithm M depends on the choice of the query $\widehat{q}$ used by the membership oracle. If $\widehat{q}$ is the CQ with canonical database C_6, then the output is $\widehat{q}$. If $\widehat{q}$ is $\widehat{q}$:- $R(x, y), R(y, z)$, the output also is $\widehat{q}$. If $\widehat{q}$ is C_n with n a multiple of 6, the output is the CQ with canonical database C_6.*

Algorithm B. We finally introduce Algorithm B which implements in a straightforward way the bounded fitting approach proposed in [16]. It is based on a size-bounded version of the fitting construction problem that also incorporates fitting existence. This is again defined relative to a query language $\mathcal{Q}$:

Size-Bounded Fitting Given a collection E of labeled examples and a size bound $s \in \mathbb{N}$ (in unary), construct a fitting query from $\mathcal{Q}$ of size at most s if it exists and report non-existence otherwise.

Size-bounded fitting tends to be of significantly lower computational complexity than fitting construction and existence without a size bound, Section 4 has details. Algorithm B calls an algorithm for the size-bounded fitting problem on E with increasing size bounds, see Algorithm 3.3. More details on algorithms for the size-bounded fitting problem are given in Section 4.

EXAMPLE 3.4. *Consider once more the collection of labeled examples $E = (E^+, E^-)$ from Example 3.1. Algorithm B outputs a fitting CQ of smallest size. In this case, there is a unique such CQ, which is q :- R(x,y), R(y,z).*

4. RUNNING TIME AND SIZE BOUNDS

We discuss the computational complexity of fitting construction and fitting existence and, closely related, the size that the smallest CQ that fits a collection of examples may have in the worst case.

Fundamental Considerations. It was shown in [40] that the smallest fitting CQ for a given collection of labeled examples is in general of size exponential in the size of the examples. We illustrate this with an example.

EXAMPLE 4.1. *Let p_i denote the i-th prime number (where $p_1 = 2$). For $n \geq 1$, let $E_n = (E^+, E^-)$ be the collection of labeled examples with $E^+ = \{C_{p_2}, \ldots, C_{p_{n+1}}\}$, and $E^- = \{C_2\}$. By the prime number theorem, the total size of the examples in E_n is polynomial in n. However, every fitting CQ has size greater than 2^n. More precisely, let $k = \Pi_{i=2}^{n+1} p_i$. The canonical CQ of C_k fits E_n and no smaller fitting CQ exists. Indeed, every fitting CQ q must contain a directed cycle in order to fit the negative example, and, in order to fit to the positive examples, the length of each directed cycle in q must be a multiple of p_i for $2 \leq i \leq n+1$, and hence a multiple of k.*

It follows immediately that the fitting problem cannot be solved by an algorithm with sub-exponential running time, simply because it does in general not have enough time to output a fitting. Less trivially, the inherent complexity of the problem also precludes the existence of such an algorithm. This is witnessed by the following.

THEOREM 4.2. *The fitting existence problem for CQs is coNExpTime-complete.*

This result was first shown in [40] and later improved to hold for a fixed finite schema in [12]. The upper bound is actually shown by running Algorithm P, whose output q is guaranteed to fit all positive examples, and then checking in coNP whether q fits the negative examples. One can show that this is the case if and only if a fitting exists.

Theorem 4.2 does not preclude the possibility of a fitting algorithm that runs in time polynomial in the size of the input plus the size of the smallest fitting CQ. Recall that we call such an algorithm *weakly polynomial*. Unfortunately, it follows from the results in [17] that there is no weakly polynomial fitting algorithm for CQs unless P=NP. In fact, it is shown in [17] that there is a class of examples $\mathcal{E}$ such that

(i) fitting existence for CQs is NP-hard on collections of examples from $\mathcal{E}$;

(ii) if there exists any fitting for such a collection E, then there is one of size bounded by $p(||E||)$ for some polynomial p;

(iii) CQs can be evaluated in polynomial time (in combined complexity) on $\mathcal{E}$.

Any weakly polynomial fitting algorithm for CQs would allow us to decide the problem in (i) in polynomial time, thus showing P=NP, as follows. Given a collection E of examples from $\mathcal{E}$ that may or may not have a fitting CQ, we run the algorithm. Note that the algorithm's behavior is unspecified in the case that $\mathcal{E}$ has no fitting: it may return something else than a fitting CQ or never terminate. If the algorithm makes an output, then by (iii) we may check in polynomial time if it is a fitting CQ for E and return 'yes' or 'no' accordingly. If the algorithm exceeds the running time of $q(|E| + p(|E|))$, where q is the polynomial running time bound of the algorithm on collections of data examples that are promised to have a fitting and p the polynomial from point (ii), then we know that $\mathcal{E}$ has no fitting CQ and thus may return 'no'.

Restricted Fitting Problems. It is natural to ask whether the complexity of the fitting problem can be reduced by restricting attention to a subclass of CQs.

Fitting existence is ExpTime-complete for tree CQs [25] and thus still far from tractable. The upper bound extends to classes of CQs whose treewidth is bounded by a constant [7]. Smallest fitting tree CQs may even be of double exponential size [14]. For CQs that take the form of a path, fitting existence is still NP-complete and thus intractable [17]. In this case, however, there is always a polynomially-size fitting (if a fitting exists at all). Nevertheless, what was said above about weakly polynomial fitting algorithms applies already to these restricted classes.

Other (rather strong) types of syntactic restrictions, such as limitations on the use of existential variables, determinacy conditions pertaining to functional relations, and restricted variable depth, were proposed and studied in the literature on ILP in order to gain tractability. An overview can be found in [35].

Instead of changing the class of queries, one may also adopt other restrictions. Requiring that all examples are based on the same database instance, and even that every tuple of domain elements from that instance occur as a positive or negative example (cf. Remark 2.4), does not improve the complexity [40, 12]. An interesting variation motivated by the bounded fitting approach that underlies our Algorithm B is the size-bounded fitting problem, see Section 3. However, also this restriction does not bring tractability. It was shown in [27] that the size-bounded fitting problem for CQs is Σ_2^p-complete (over a schema with an infinite number of relation symbols of arity at most three). Even for tree CQs that take the form of a path, size-bounded fitting is still NP-complete [17].

How Our Three Algorithms Fare. Algorithm P runs in single exponential time, which is worst-case optimal in light of the above considerations. Note, however, that Algorithm P also has *best case* exponential running time which clearly makes it impractical. Of course, Algorithm P is also not weakly polynomial—we had argued above that no algorithm can be. Example 5.1 below gives a concrete family of example collections where a constant-size fitting CQ exists, but Algorithm P computes a fitting that has exponential size.

Algorithm M, which may be viewed as a refinement of Algorithm P, has single exponential running time (independently of the choice of $\widehat{q}$). In contrast to Algorithm P, however, it *is* weakly polynomial provided that the CQ $\widehat{q}$ chosen for the oracle MEMB$_{\widehat{q}}$ is the smallest fitting CQ [17]. The intuitive reason is that, after each minimization step, the size of the example e^*, which represents the current candidate query, must be bounded from above by the size of $\widehat{q}$; this is because we need at most one fact in e^* for each atom in $\widehat{q}$. Note that the result on the non-existence of weakly polynomial algorithms from above does not apply to Algorithm M because of its use of membership queries.

The worst case running time of Algorithm B is even double exponential (and it is not weakly polynomial). In fact, we have seen in Example 4.1 that smallest fitting CQs may be (single) exponentially large. Moreover, the size of the smallest fitting determines the size bound s that Algorithm B needs to solve the size-bounded fitting problem for and, as noted above, for CQs this problem is Σ_2^p-complete. Algorithm B is thus a NExpTime$^{\text{NP}}$ algorithm and has the highest worst-case complexity among our three algorithms. Interestingly, it is nevertheless a promising approach to the efficient implementation of the fitting problem. This is based on the expectation that, in practical cases, the size of the smallest fitting query tends to not be excessively large.

An implementation of Algorithm B ("bounded fitting") as well as practical experiments have been presented in [16] for the case of tree CQs. In that case, the size-bounded fitting problem can be translated in a natural way into the satisfiability problem of propositional logic which enables an efficient implementation based on SAT solvers. The SPELL system described in [16] shows competitive performance and often outperforms state-of-the-art systems based on refinement operators (which are discussed in Section 9). For unrestricted CQs, one may attempt to replace the SAT solver with a system for answer set programming or disjunctive logic programming, but we are not aware that this has been done in practice. As we will see in Section 7, Algorithm B has the additional advantage of constructing fittings that generalize well to unseen examples.

5. SUCCINCT FITTINGS

One desirable property of a fitting algorithm is *succinctness*: for many applications, one wants to find a fitting query of small size. We saw in Example 4.1 that the smallest fitting CQ for a given collection of labeled examples may be exponentially large in the worst case. Therefore, the best one can hope for is to produce a fitting query of size not much larger than that of the smallest fitting CQ. For fitting algorithms with access to a membership oracle $MEMB_{\widehat{q}}$, even this is to much: we can only hope to produce a fitting CQ of size not much larger than that of (the smallest query equivalent to) $\widehat{q}$. Intuitively, this is because the membership oracle 'guides' the fitting algorithm to $\widehat{q}$ and not to a smaller fitting CQ.

Let us consider Algorithm P (which does not use a membership oracle). The following example shows that it may produce a fitting CQ that is much larger than a fitting CQ of minimal size. In fact, the size of the CQ produced by Algorithm P cannot be uniformly bounded by any function in the size of the smallest fitting CQ.

EXAMPLE 5.1. *Consider again Example 4.1. We modify the example slightly: for $n > 0$, let $E_n = (E^+, E^-)$ where $E^+ = \{C_{p_2}, \dots, C_{p_{n+1}}\}$ (as before), and $E^- = \emptyset$. On input E_n, Algorithm P outputs a CQ of size $\Theta(\Pi_{i=2}^{n+1} p_i)$ (and not equivalent to any smaller CQ), whereas there is a single (and thus constant size) CQ that fits E_n for all n, namely q :- $R(x, y)$.*

As we have just seen, Algorithm P does not even produce a fitting CQ of "near-minimal" size. In the context of PAC learning as discussed in Section 7, fittings of near-minimal size play an important role. It will be beneficial to formalize this notion already here:

Occam Property A fitting algorithm has the *Occam property* if the following holds for some $\alpha \in [0, 1)$ and polynomial p: if the input is a collection of examples labeled according to a "target CQ" q_t, then the output is a fitting CQ of size at most $|E|^\alpha \cdot p(|q_t|)$. If $\alpha = 0$, we say that the fitting algorithm has the *strong Occam property.*[2]

In other words, when a fitting algorithm has the Occam property, it outputs a CQ whose size depends polynomially on the size of the target CQ and sublinearly on the number of the examples (if at all). The above definition of the Occam property is designed to apply both to ordinary fitting algorithms and to fitting algorithms that use a membership oracle. In the latter case, the target CQ q_t in the above definition is required to be the CQ used by the oracle. In the former case, we can always assume without loss of generality that q_t is a fitting CQ of minimal size. Consequently, a fitting algorithm without membership oracle has the Occam property if and only if it outputs a CQ of size at most $|E|^\alpha \cdot p(n)$, where n is the size of the *smallest CQ that fits*. The latter is indeed the standard condition used to define Occam algorithms in the literature on computational learning theory (where Occam algorithms with membership oracles are typically not considered).

[2]The terminology "Occam property" is not entirely standard. In the literature, it is more common to talk about an *Occam algorithm*, meaning a fitting algorithm that has the Occam property *and* is weakly polynomial. We already saw that there is no weakly polynomial fitting algorithm for CQs (without oracle) and thus in particular there is no such Occam algorithm. By decoupling the size of the output from the running time of the algorithm, we can more easily acknowledge that there are super-polynomial-time fitting algorithms with the Occam property (as we will see below).

How Our Three Algorithms Fare. Example 5.1 shows that Algorithm P does not have the Occam property. As we will see, this is an instance of a general phenomenon related to extremal fitting CQs, see Example 6.2.

Algorithm B *does* have the strong Occam property: indeed, it is immediate from the definition of the algorithm that the output CQ is a fitting CQ of minimal size. Note that this does not depend on the fact that Algorithm B increases the size bound by 1 in each iteration. In fact, under the scheme $s = 1, 2, 4, 8$ it still has the strong Occam property (with $p(x) = 2x$).

For Algorithm M, it was shown in [17] that it outputs a CQ whose size is bounded by the size of the query $\widehat{q}$ used by the membership oracle. Therefore, Algorithm M has the strong Occam property.

6. EXTREMAL FITTINGS

We have already seen that there may be several non-equivalent fitting CQs for the same collection of data examples, and in fact it is easy to see that there may be infinitely many. As was observed in [14], the fitting CQs always form a *convex set*. More precisely, whenever two queries q_1, q_2 fit a set of labeled examples, the same holds for every query q with $q_1 \subseteq q \subseteq q_2$. Recall that $\subseteq$ denotes the relation of query containment, i.e., $q_1 \subseteq q_2$ means that $q_1(I) \subseteq q_2(I)$ for all instances I. The maximal elements of the convex set of fitting CQs can be viewed as "most-general" fitting CQs while minimal elements can be viewed as "most-specific" fitting CQs. We refer to these, collectively, as "extremal" fitting CQs. When they exist, they can thus be viewed as demarcating the entire set of fitting CQs, in the spirit of the version-space representation theorem used in machine learning [34, Chapter 2.5]. In applications such as ML feature engineering over relational data [30, 8], extremal fitting CQs are particularly natural candidates to consider as features [14].

Most-Specific Fitting CQs. There are two natural ways to define "most-specific fitting CQs" for a collection of labeled examples E: a CQ q is a

- *strongly most-specific fitting* CQ for E if q fits E and for every CQ q' that fits E, we have $q \subseteq q'$;
- *weakly most-specific fitting* CQ for E if q fits E and for every CQ q' that fits E, $q' \subseteq q$ implies $q \equiv q'$.

There can clearly be at most one strongly most-specific fitting CQ up to equivalence, for any collection of labeled examples. In contrast, the existence of multiple weakly most-specific fitting CQs is not excluded a priori. It turns out, however, that also weakly most-specific fitting CQs are unique and in fact the two notions coincide and are characterized by the product of the positive examples.

THEOREM 6.1. *For all CQs q and collections of labeled examples $E = (E^+, E^-)$, the following are equivalent:*

1. *q is a strongly most-specific fitting for E,*
2. *q is a weakly most-specific fitting for E,*
3. *q fits E and is equivalent to the canonical CQ of $\Pi_{e \in E^+}(e)$ (which must then be well-defined).*

Since Algorithm P computes the canonical CQ of the direct product of the positive examples, it in fact produces a most-specific fitting CQ. This also means that most-specific fitting CQs always exist (if any fitting CQ exists). In contrast, Algorithm M and Algorithm B do *not* in general produce a most-specific fitting:

EXAMPLE 6.2. *Recall that for the collections E_n of labeled examples from Example 5.1, there exists a fitting CQ with a single atom, which is thus output by Algorithms M and B. In contrast, we had seen in Example 5.1 that Algorithm P produces a CQ of exponential size. We claim that, in fact,* every *most-specific fitting CQ q for E_n must be of size at least $k = \Pi_{i=2}^{n+1} p_i$. Indeed, let q be any most-specific fitting CQ q and let q' be the query that expresses the existence of a directed cycle of length k. Since q fits E_n and q' is a (strongly) most-specific fitting CQ, it follows that $q' \subseteq q$. Consequently, q' must contain a directed cycle. However, as we argued in Example 4.1, the length of any such directed cycle must be a multiple of k. Therefore, q must be of size at least k.*

More generally, the example shows that no fitting algorithm with the Occam property will always output a most-specific fitting CQ.

Most-General Fitting CQs. For most-general fittings, the story gets more complicated. There are again two natural ways to define "most-general fitting CQs" for a collection of labeled examples E: a CQ q is a

- *strongly most-general fitting* CQ for E if q fits E and for every CQ q' that fits E, we have $q' \subseteq q$;
- *weakly most-general fitting* CQ for E if q fits E and for every CQ q' that fits E, $q \subseteq q'$ implies $q \equiv q'$.

This time, however, the two notions do not coincide. While every strongly most-general fitting CQ is clearly also weakly most-general, the converse fails:

EXAMPLE 6.3. *Consider a schema consisting of three unary relation symbols P_1, P_2, P_3. Let $E = (E^+, E^-)$ where $E^+ = \emptyset$ and E^- consists of the single-fact instance $\{P_1(a)\}$. Then q :- $P_2(x)$ and q' :- $P_3(x)$ are weakly most-general fitting CQs for E and there is no strongly most-general fitting CQ.*

Another natural variation is the following [14]:

- a finite set of CQs $\{q_1, \dots, q_n\}$ is a *basis of most-general fitting CQs* for E if each q_i fits E and for all CQs q' that fit E, we have $q' \subseteq q_i$ for some $i \leq n$.

It is easy to see that a strongly most-general fitting CQ is simply a fitting CQ that forms a basis of size 1, and that every member of a *minimal* basis of most-general fitting CQs is a weakly most-general fitting.

Unlike for most-specific fitting CQs, the existence of a fitting CQ does not, in general, imply the existence of a most-general fitting CQ. For the strong version, this is shown by Example 6.3. For the weak version, we consider the following example [14].

EXAMPLE 6.4. *Let $E = (E^+, E^-)$ be the collection of labeled examples with $E^+ = \{C_1\}$ and $E^- = \{C_2\}$. Clearly, the Boolean CQ q_0 :- $R(x,x)$ fits E. It follows from results in [14] that no weakly most-general fitting CQ exists for E. Intuitively, this is because a CQ q fits E if and only if its canonical instance, viewed as a graph, is not two-colorable. A graph is two-colorable if and only if it does not contain a cycle of odd length. Thus, each fitting CQ for E must contain a cycle of odd length, and for each such CQ q one can construct a strictly more general fitting CQ q' by increasing the length of this cycle.*

Thus, one cannot reasonably require a fitting algorithm to always output a most-general fitting CQ. Still, one may wish for a fitting algorithm that, on inputs for which a most-general fitting CQ exists, produces such a CQ. The "most-general fitting" here may refer to *weakly* most-general fitting CQs or to *strongly* most-general ones, so there are two variants of this requirement. It is not hard to see that the first variant implies the second.

As it turns out, even if we restrict attention to instances of the fitting problem for which a strongly most-general fitting CQ exists, Algorithm P, M and B are not guaranteed to produce one. The following example shows this for Algorithm P.

EXAMPLE 6.5. *For $n > 1$, let T_n be the database instance with $adom(T_n) = \{a_1, \ldots, a_n\}$ consisting of all facts of the form $R(a_i, a_j)$ with $i < j$. Let $E_n = (E^+, E^-)$ where $E^+ = \{C_1\}$ and $E^- = \{T_n\}$. The following two queries both fit E_n:*

$$\begin{aligned} q &\text{ :- } R(x,x) \\ q' &\text{ :- } R(x_1,x_2), R(x_2,x_3), \ldots, R(x_n,x_{n+1}). \end{aligned}$$

Observe that $q \subsetneq q'$. In fact, q is the fitting CQ for E_n of smallest size while it can be shown that q' is the strongly most-general fitting CQ for E_n.

It is easy to see that Algorithm P outputs q on input E. Therefore, Algorithm P does not, in general, output a strongly most-general fitting CQ when such a CQ exists.

More generally, the example shows that no fitting algorithm with the Occam property will always return a strongly most-general fitting CQ whenever it exists. Since Algorithms B and M have the Occam property, it follows that they do not, in general, output a strongly most-general fitting CQ when such a CQ exists.

Algorithms for Most-General Fitting CQs. Since none of our three fitting algorithms produces most-general fitting CQs even when they exist, it is natural to ask whether there are fitting algorithms with this property. The existence problem for weakly/strongly most-general fitting CQs and the problem of constructing them, when they exist, were studied extensively in [14]. Both of these problems turn out to be decidable and we briefly review the core insights underlying the algorithms.

Weakly most-general fitting CQs can be characterized in terms of *frontiers* [14]. A *frontier* for a CQ q is a finite set of CQs $F(q) = \{q_1, \ldots, q_n\}$ with the property that $q \subsetneq q_i$ for all $i \leq n$, and for all CQs q', if $q \subsetneq q'$ then $q_i \subseteq q'$ for some $i \leq n$. Thus, a frontier for a CQ q is a finite complete set of minimal generalizations of q.

THEOREM 6.6 ([14]). *For all queries q and collections of labeled examples E, the following are equivalent:*

1. *q is a weakly most-general fitting CQ for E,*
2. *q fits E, q has a frontier $F(q) = \{q_1, \ldots, q_n\}$ and no $q_i \in F(q)$ fits E.*

This characterization, together with known results regarding the existence of frontiers for CQs [24, 13], was used in [14] to obtain effective algorithms for the existence, verification, and construction problem for weakly most-general fitting CQs. Regarding the computational complexity, we only mention here that the existence problem is ExpTime-complete [14]. It is worth to point out, however, that the known lower bound only applies when we do not require that the collection of labeled data examples E given as an input has a fitting CQ.

Theorem 6.6 also sheds light on the shape of weakly most-general fitting CQs. In fact, it is known that any CQ that has a frontier must be *c-acyclic* [1, 24, 13] which means that every cycle in the incidence graph of the CQ contains an answer variable. By Theorem 6.6, the same is true for weakly most-general fitting CQs. This is crucially exploited by the algorithms in [14].

Strongly most-general fitting CQs and, more generally, finite bases of most-general fitting CQs turn out to be closely related to *homomorphism dualities*, a fundamental concept that originates from combinatorial graph theory and has found diverse applications in different areas, including the study of constraint satisfaction problems, database theory, and knowledge representation.

With $adom(I)$, we denote the set of values used in database instance I. Recall that a homomorphism $h\colon I \to J$ from instance I to instance J (over the same schema) is a function $h\colon adom(I) \to adom(J)$ such that the h-image of every fact of I is a fact of J. We write $I \to J$ to indicate the existence of a homomorphism from I to J. A *homomorphism duality* is a pair of finite sets of instances $(\mathcal{F}, \mathcal{D})$ such that for all instances I, $F \to I$ for some $F \in \mathcal{F}$ iff $I \not\to D$ for all $D \in \mathcal{D}$. This notion can be further refined by relativizing it: a pair $(\mathcal{F}, \mathcal{D})$ being a homomorphism duality *relative to* an instance J is defined in exactly the same way except that only instances I are considered that satisfy $I \rightarrow J$.

THEOREM 6.7 ([14]). *For all Boolean CQs $q_1, \ldots, q_n$ and collections of labeled examples E, the following are equivalent:*

1. *$\{q_1, \ldots, q_n\}$ is a basis of most-general fitting CQs for E*

2. *each q_i fits E and $(\{I_{q_1}, \ldots, I_{q_n}\}, E^-)$ is a homomorphism duality relative to $\Pi_{e \in E^+}(e)$.*

This characterization (and an extension of it for non-Boolean CQs), together with known results on the existence of homomorphism dualities, was used in [14] to obtain effective algorithms for the existence, verification, and construction of bases of most-general fitting CQs. Regarding the computational complexity, we only mention here that the existence problem is NExpTime-complete [14]. The lower bound applies even if the collection of labeled data examples E given as an input is promised to have a fitting CQ. Note that since every member of a minimal basis of most-general fittings CQs is a weakly most-general fitting, it must be c-acyclic.

7. GENERALIZATION

By definition, a fitting algorithm constructs a CQ that fits the labeled examples in its input. Ideally, we would like the constructed CQ to correctly predict the label also of *unseen* examples. This can only be expected if those examples are drawn from the same probability distribution (over the space of all examples) as the input examples, and labeled according to the same target CQ. If it is the case, then we can legitimately say that the fitting generalizes from the input. This is formally captured by the framework of *PAC (Probably Approximately Correct)* learning [39]. To give precise definitions, we first introduce some terminology and notation. An *example distribution* is a probability distribution D over the space of all examples (for some fixed schema and arity). Given CQs q, q_t and an example distribution D,

$$\text{error}_{D,q_t}(q) = \Pr_{(I,\mathbf{a}) \in D}(\mathbf{a} \in q(I) \triangle q_t(I))$$

is the expected error of q relative to q_t and D where $\triangle$ denotes symmetry difference. Hence, $\text{error}_{D,q_t}(q)$ is the probability that q disagrees with q_t on any example drawn at random from the example distribution D.

(Polynomial) PAC Property A fitting algorithm has the (polynomial) PAC property if there is a (polynomial) function $f(\cdot,\cdot,\cdot,\cdot)$ such that for all CQs q_t, $\delta, \varepsilon \in (0,1)$, $m \in \mathbb{N}$ and probability distributions D over examples of size at most m, if the input consists of at least $f(1/\delta, 1/\varepsilon, |q_t|, m)$ many examples drawn from D that are labeled according to q_t, then with probability at least $1-\delta$, the algorithm outputs a CQ q with $\text{error}_{q_t,D}(q) < \varepsilon$.[3]

Other common definitions of efficient PAC learning do not even demand that the learning algorithm produces a fitting query, but require that the algorithm is weakly polynomial. However, it is known that there is no such efficient PAC learning algorithm for CQs. A detailed discussion can be found in [17], also for subclasses of CQs such as tree CQs and path-shaped CQs which are not efficiently PAC learnable either. This is closely related to the fact that there is no weakly polynomial fitting algorithm for (these subclasses of) CQs, see Section 4.

A fundamental result in computational learning theory states that every fitting algorithm with the Occam property also has the polynomial PAC property, first shown in [9]. This result is usually stated only for fitting algorithms without a membership oracle, but the same holds in the presence of such an oracle. This relies on the fact that, in Section 5, we defined the Occam property in terms of a target CQ rather than a smallest fitting CQ.

Since Algorithm M has the Occam property, we can conclude that it has the polynomial PAC property. The same applies to Algorithm B. Algorithms P, on the other hand, lacks the polynomial PAC property. In fact:

THEOREM 7.1 ([16, 15]). *Let A be a fitting algorithm that either (i) always produces a most-specific fitting or (ii) produces a strongly most-general fitting whenever it exists. Then A lacks the polynomial PAC property.*

The intuitive reason behind Theorem 7.1 is that extremal fittings tend to overfit to the input examples. Most-specific fittings focus too much on the positive examples in the input and tend to incorrectly predict the label of unseen positive examples. Similarly, most-general fittings focus too much on the negative examples in the input. It is worth contrasting Theorem 7.1 with the result from [5, 29] that, for concept classes with finite VC dimension that are intersection-closed, fitting algorithms that produce most-specific fittings have the polynomial PAC property.

[3]We deviate here from standard textbook definitions of the PAC model by including the bound m on the size of examples as an argument to the sample complexity function f. Most concept classes studied in the computational learning theory literature have an example space that consists of examples of bounded size. The example size can then be treated as a constant and thus m can be omitted as an argument to the function f. In contrast, fitting algorithms for CQs can receive arbitrarily large database instances as inputs. The present definition of the PAC property, following [35], allows the number of examples provided to the fitting algorithm to depend on the example size. That is, if the input contains large examples, the fitting algorithm is accordingly given access to more labeled examples. This is not needed for the positive results mentioned below, but it makes the negative results stronger.

8. COMPLETENESS FOR DESIGN

One application of fitting algorithms is in the context of example-based specification ("query by example"): rather than writing a formal specification, the user provides data examples, and the system infers a query through the use of a fitting algorithm. The premise of this approach, which traces back to [41], is that the user has a good grasp of the desired behavior of the query that they are trying to construct, but not necessarily of the query language. In such a setting, it is desirable that the user is indeed able to obtain their intended query as long as they provide a sufficiently comprehensive (finite) set of examples. Whether this is the case, in general, depends on the fitting algorithm, and this is formalized by the following property:

Completeness for Design A fitting algorithm is *complete for design* if for every CQ q, there exists a collection of labeled examples E such that, on input E, the fitting algorithm produces a CQ equivalent to q.

If a fitting algorithm is not complete for design, there are CQs q for which the algorithm will simply never get it right, no matter how many examples are provided.

It is not difficult to see that every fitting algorithm that produces a most-specific fitting CQ, is complete for design. Indeed, it suffices to pick any collection of examples (labeled according to the query q) that includes the canonical example of q. In particular, this shows that Algorithm P is complete for design.

Similarly, it can be shown that every fitting algorithm with the strong Occam property is complete for design. Indeed, let q be any CQ, and E be any collection of examples, labeled according to q, that includes, for every CQ q' of size at most $p(|q|)$ not equivalent to q a labeled examples that q fits but q' does not. The fitting algorithm, on input E, is guaranteed to produce a fitting CQ of size at most $p(|q|)$, which therefore, by construction, must be equivalent to q. Since Algorithm M and Algorithm B have the Occam property with $\alpha = 0$, it follows that both are complete for design.

Completeness for design is also related to the notion of unique characterizability [13]. More precisely, if a class of queries admits unique characterizations (which means that every query is uniquely characterized by a finite set of data examples), then every fitting algorithm for this class is complete for design. There is no implication in the opposite direction. In fact, CQs are not uniquely characterizable (cf. [13]), even though our fitting algorithms are complete for design.

9. REFINEMENT-BASED APPROACHES

A different approach to the fitting problem has been taken in the field of *inductive logic programming (ILP)* which studies the following abstract problem [35]:

> Given a background theory $\mathcal{B}$ and positive and negative examples, find a theory Σ such that $\mathcal{B} \cup \Sigma$ entails all positive examples and none of the negative examples.

Traditionally, this problem is studied for the language of first-order clauses. Thus, the background theory $\mathcal{B}$ is a finite set of clauses, the examples are clauses, and the sought theory can also be a set of clauses. Note that this a very rich problem setting since first-order clauses include, e.g., all Datalog programs. The CQ fitting problem can be viewed as a special case when all examples in the input (E^+, E^-) share the same database I: the background theory is $\mathcal{B} = I$, there is a positive example $R(\mathbf{a})$ for each $(I, \mathbf{a}) \in E^+$, a negative example $R(\mathbf{a})$ for each $(I, \mathbf{a}) \in E^-$, where R is a fixed relation symbol that does not appear in I, and Σ is restricted to be a single non-recursive Horn clause with head $R(\mathbf{x})$.

Most ILP algorithms conform to a common scheme [35]: start with some initial theory Σ, and, while $\mathcal{B} \cup \Sigma$ is not as required, iteratively adapt Σ as follows:

- if $\mathcal{B} \cup \Sigma$ is *too strong* (entails a negative example), generalize Σ, and
- if $\mathcal{B} \cup \Sigma$ is *too weak* (does not entail a positive example), specialize Σ.

Initially, generalization was done by dropping a clause from Σ while specialization was done by adding a clause. However, it was observed that the induced changes to Σ are too coarse. To address this, Ehud Shapiro introduced in a seminal paper *refinement operators*, which specialize, respectively generalize a given clause [37]. Applied to the task of finding a fitting CQ (or single Horn clause), this amounts to navigating the *containment lattice* of CQs, whose investigation goes back at least to the 1970s [36].

Formally, an *upward (resp., downward) refinement operator* is a function $\rho : \text{CQ} \to 2^{\text{CQ}}$ such that $q \subseteq q'$ (resp., $q' \subseteq q$) for every $q \in \text{CQ}$ and $q' \in \rho(q)$. Thus, an upward refinement operator returns a set of more general queries, while a downward refinement operator returns a set of more specific queries. Intuitively, a refinement operator induces a graph whose vertices are CQs (equivalent CQs form a single vertex) and in which there is an edge from q to q' iff $q' \in \rho(q)$. ILP style fitting algorithms will then search this graph using some strategy. Algorithm 9.1 depicts a template for such an algorithm based on an upward refinement operator ρ and a prioritization strategy S that determines which query to consider next in search. It starts with the most specific query possible and maintains a priority queue of queries to be visited. In each round it selects the query with the highest priority and, if it does not fit, adds its refinements to the queue, prioritized by S. Natural prioritization strategies include breadth-first search (query gets lower priority than all previously seen queries) and accuracy-based

Algorithm 9.1: Algorithm R, parameterized by refinement operator ρ and prioritization strategy S

```
Input  : collection $(E^+, E^-)$ of labeled examples
Output : fitting CQ or "None exists"
1 $q_0$ := canonical CQ of $e_\top^k$ for $k$ the arity of $(E^+, E^-)$;
2 PriorityQueue pq := $\{q_0\}$;
3 while not pq.isEmpty() do
4     $q$ := pq.pop();
5     if $q$ fits $(E^+, E^-)$ then return $q$;
6     Insert every $p \in \rho(q)$ into pq prioritized with $S$
7 return "None exists"
```

strategies (the priority is the accuracy of the considered query over (E^+, E^-)). There is a natural counterpart of Algorithm 9.1 based on downward refinement operators.

It has been observed that, to make Algorithm 9.1 a complete and terminating algorithm for the CQ fitting problem, the refinement operator ρ has to be ideal, which we define next. An upward refinement operator ρ is called *proper* if $q' \not\subseteq q$ for every $q' \in \rho(q)$ and all $q \in$ CQ, *finite* if $\rho(q)$ is finite for every $q \in$ CQ, and *complete* if for every $q_1, q_2 \in$ CQ with $q_1 \subseteq q_2$, there is a sequence $p_1, \ldots, p_n$ of CQs with $p_1 = q_1$, $p_n = q_n$, and $p_{i+1} \in \rho(p_i)$ for all i with $1 \leq i < n$. It is *ideal* if it is proper, finite, and complete. Ideal downward refinement operators are defined similarly.

There is an intimate connection between ideal refinement operators and frontiers as defined in Section 6. We call them *upward frontiers* here and use the dual notion of *downward frontiers*, defined as expected. Frontiers are also called *covers* in the ILP literature. It has been shown that the existence of finite frontiers is a necessary condition for the existence of ideal refinement operators, and that finite frontiers do not always exist, both in the upward and the downward case [19, 35]. This holds in particular for CQs, and even more so for the broader classes of theories Σ that ILP systems aim to support. Thus, in practice one has to compromise, either by dropping one of the three properties (finiteness, completeness, properness), or by restricting the query class. Most ILP systems use an incomplete refinement operator together with heuristics [35, 21]. As our focus is on complete algorithms, we discuss below restrictions of the query class.

A class of queries which enjoys finite frontiers is the class of tree CQs. Indeed, every tree CQ has a polynomially sized upward frontier [6] and an exponentially sized downward frontier [32, 31]. Moreover, for every q, q' with $q \subseteq q'$, there is only a finite number of queries p with $q \subseteq p \subseteq q'$ [31]. Thus, the function returning the frontier of a query *is* an ideal refinement operator, both in the upward and downward case. Other classes of CQs that have finite frontiers were studied in [13, 26]. It is not known whether they admit ideal refinement operators.

How Algorithm R Fares. A general classification in terms of our desired properties is difficult, since the behavior depends on the refinement operator and prioritization strategy used. We discuss only preliminary observations for tree CQs. On the positive side, if the downward frontier is used as a refinement operator, combined with breadth-first search, the downward version of Algorithm 9.1 will always return a weakly most-general fitting (when it exists) [16]. Of course, this also means, by Theorem 7.1, that it does not have the PAC property. This can be fixed by extending the refinement operator in a suitable way to achieve that the resulting algorithm has the Occam property [16]. On the negative side, the length of refinement paths along upward frontiers is not bounded by an elementary function [31], which compromises efficiency for the upward version of Algorithm 9.1.

An ideal downward refinement operator for tree CQs developed in [32] has been implemented in the ELTL incarnation of the DL Learner suite [11]. Since the prioritization strategy used there is quite involved, it is hard to analyze whether ELTL satisfies our desired properties. Experiments from [16] show that ELTL generalizes well to unseen examples, which might be explained by the fact that its prioritization strategy takes the query size into account and gears search towards smaller CQs.

10. SUMMARY AND OUTLOOK

We identified a list of desirable properties of fitting algorithms and their structural interactions (cf. Figure 1), and used them to compare three fitting algorithms for CQs (cf. Table 2). As mentioned in Section 4, Algorithm B was successfully implemented and shown to perform competitively for the special case of tree CQs [16].

Our general motivation comes from the development of *interactive, example-aided methodologies for the synthesis, refinement, and debugging of database queries.* The fitting problem is only one facet of this broader topic. Other facets that require further study include example generation and example-based query refinement.

Extending the scope of our analysis to other query languages remains future work. A detailed analysis of extremal fitting problems for UCQs can be found in [14].

We only considered *exact* fittings. When these are not guaranteed to exist, the fitting problem is perhaps more naturally viewed as a multi-objective optimization problem (with degree-of-fitting as one of the objectives). This is future work but see [8, 28, 18] for some related work.

Acknowledgements. B. ten Cate is supported by the European Union's Horizon 2020 research and innovation programme (MSCA-101031081). C. Lutz is supported by DAAD project 57616814 (SECAI, School of Embedded Composite AI) as part of the program Konrad Zuse Schools of Excellence in Artificial Intelligence.

11. REFERENCES

[1] B. Alexe, B. t. Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, 2011.

[2] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *Proc. of SIGMOD*, pages 133–144, 2011.

[3] D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, Apr. 1988.

[4] M. Arenas, G. I. Diaz, and E. V. Kostylev. Reverse engineering sparql queries. In *Proc. of WWW*, page 239–249, 2016.

[5] P. Auer and R. Ortner. A new pac bound for intersection-closed concept classes. *Machine Learning*, 66:151–163, 2004.

[6] F. Baader, F. Kriegel, A. Nuradiansyah, and R. Peñaloza. Making repairs in description logics more gentle. In *Proc. KR*, pages 319–328, 2018.

[7] P. Barceló and M. Romero. The complexity of reverse engineering problems for conjunctive queries. In *Proc. of ICDT*, pages 7:1–7:17, 2017.

[8] P. Barceló, A. Baumgartner, V. Dalmau, and B. Kimelfeld. Regularizing conjunctive features for classification. *J. Comput. Syst. Sci.*, 119:97–124, 2021.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[10] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *Proc. of EDBT*, pages 109–120, 2015.

[11] L. Bühmann, J. Lehmann, and P. Westphal. DL-Learner - A framework for inductive learning on the semantic web. *J. Web Semant.*, 39:15–24, 2016.

[12] B. ten Cate and V. Dalmau. The product homomorphism problem and applications. In *Proc. of ICDT*, pages 161–176, 2015.

[13] B. ten Cate and V. Dalmau. Conjunctive queries: Unique characterizations and exact learnability. *ACM Trans. Database Syst.*, 47(4):14:1–14:41, 2022.

[14] B. ten Cate, V. Dalmau, M. Funk, and C. Lutz. Extremal fitting problems for conjunctive queries. In *Proc. of PODS*, 2023.

[15] B. ten Cate, M. Funk, J. C. Jung, and C. Lutz. Extremal fitting CQs do not generalize. *CoRR*, abs/2312.03407, 2023.

[16] B. ten Cate, M. Funk, J. C. Jung, and C. Lutz. SAT-based PAC learning of description logic concepts. In *Proc. IJCAI*, pages 3347–3355, 2023.

[17] B. ten Cate, M. Funk, J. C. Jung, and C. Lutz. On the non-efficient PAC learnability of conjunctive queries. *Inf. Process. Lett.*, 183:106431, 2024.

[18] B. ten Cate, P. G. Kolaitis, K. Qian, and W.-C. Tan. Approximation algorithms for schema-mapping discovery from data examples. *ACM Trans. Database Syst.*, 42(2):12:1–12:41, 2017.

[19] P. R. J. Laag van der Laag and S. Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In *Proc. of ECML*, pages 307–322, 1994.

[20] S. Cohen and Y. Y. Weiss. The complexity of learning tree patterns from example graphs. *ACM Trans. Database Syst.*, 41(2):14:1–14:44, 2016.

[21] A. Cropper, S. Dumančič, R. Evans, and S. H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022.

[22] V. Filipetto. Constructing queries from data examples, 2022. MSc Thesis. University of Amsterdam.

[23] G. Fletcher, M. Gyssens, J. Paredaens, and D. Van Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Transactions on Knowledge and Data Engineering*, 21:939–942, 2009.

[24] J. Foniok, J. Nesetril, and C. Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.*, 29(4):881–899, 2008.

[25] M. Funk, J. Jung, C. Lutz, H. Pulcini, and F. Wolter. Learning description logic concepts: When can positive and negative examples be separated? In *Proc. of IJCAI*, pages 1682–1688, 2019.

[26] M. Funk, J. C. Jung, and C. Lutz. Frontiers and exact learning of $\mathcal{ELI}$ queries under DL-Lite ontologies. In *Proc. IJCAI*, pages 2627–2633, 2022.

[27] G. Gottlob, N. Leone, and F. Scarcello. On the complexity of some inductive logic programming problems. *New Generation Comput.*, 17(1):53–75, 1999.

[28] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2):6:1–6:37, 2010.

[29] S. Hanneke. Refined error bounds for several learning algorithms. *J. Mach. Learn. Res.*, 17:1–55, 2016.

[30] B. Kimelfeld and C. Ré. A relational framework for classifier engineering. *ACM SIGMOD Record*, 47:6–13, 2018.

[31] F. Kriegel. Navigating the $\mathcal{EL}$ subsumption hierarchy. In *Proc. of DL*, 2021.

[32] J. Lehmann and C. Haase. Ideal downward refinement in the $\mathcal{EL}$ description logic. In *Proc. of ILP*, pages 73–87, 2009.

[33] H. Li, C.-Y. Chan, and D. Maier. Query from examples: An iterative, data-driven approach to query construction. *Proc. VLDB Endow.*, 8(13):2158–2169, 2015.

[34] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[35] S. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997.

[36] G. Plotkin. Lattice theoretic properties of subsumption. Technical report, Edinburgh University, Dept. of Machine Intelligence and Perception, 1970.

[37] E. Y. Shapiro. An algorithm that infers theories from facts. In P. J. Hayes, editor, *Proc. of IJCAI*, pages 446–451, 1981.

[38] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, 2014.

[39] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27:1134–1142, 1984.

[40] R. Willard. Testing expressibility is hard. In *Proc. of CP*, pages 9–23, 2010.

[41] M. M. Zloof. Query by example. In *Proc. of AFIPS NCC*, pages 431–438. AFIPS Press, May 1975.