Complexity of Relational Query Synthesis

Aalok Thakkar, Rajeev Alur, Mayur Naik
University of Pennsylvania, Philadelphia, USA
{athakkar,alur,mhnaik}@cis.upenn.edu

1 INTRODUCTION

The synthesis of relational queries from input-output examples has been studied in the context of inductive logic programming [2, 4, 6–8], program synthesis [5, 9, 11–14], and neural learning [10]. It is a challenging problem, and analyzing the computational complexity of the problem for even restricted fragments can significantly impact the development of synthesis tools.

For instance, several tools use language biases such as mode declarations, templates, meta-rules, and candidate rules to constraint the space of candidate programs [4, 6, 7, 9, 11]. Studying the hardness of the problem can allow us to determine bounds on the language biases such that completeness of the search process is not compromised. Tools that search through an infinite space do not terminate if the instance of the synthesis problem is *unrealizable* [5, 14]. Studying the unrealizability of the problem instance can help determine when such a search is futile and should be abandoned, allowing us to give termination guarantees for these tools.

The above mentioned synthesis tools consider different fragments of relational queries. This paper focuses on a fragment that corresponds to exactly to positive Datalog [1], which supports features such as conjunction, disjunction, and recursion, and does not interpret constants. We define the formal syntax and semantics of these queries as well as the synthesis problem in Section 2.

We then establish the main contribution of this paper: Relational Query Synthesis is co-NP complete (Theorem 3.2). The key insight is the construction of a query that is polynomial in the size of the input-output examples, such that the problem instance is realizable if and only if the constructed query is consistent with the examples. This result forms Section 3.

A natural challenge is that the constructed query may overfit the training data and not generalize to unseen examples. In Section 4, we look at the problem of synthesizing minimal query for a fragment of relational queries called *union of conjunctive queries* and show that the problem is at least in Σ_3^{P}. We conclude in Section 5 with the implications of these complexity results on synthesis tools and outline some directions for future work.

2 PROBLEM FORMULATION

We begin with the syntax and semantics of relational queries and then formulate the Relational Query Synthesis Problem.

2.1 Syntax of Relational Queries

A relational query Q is a set of horn clauses. To define the syntax of rules, we first fix a set of *input* predicates, a set of invented predicates, and a set of output predicates. Each predicate R is associated with an *arity* k. A *literal*, $R(v_1, v_2, \ldots, v_k)$, consists of a k-ary predicate R with a list of k variables. Then, a rule r is of the form:

$$R_h(\vec{u}_h) \text{ :- } R_1(\vec{u}_1), R_2(\vec{u}_2), \ldots, R_n(\vec{u}_n),$$

where the single literal on the left, $R_h(\vec{u}_h)$, is the *head* of r and $R_1(\vec{u}_1)$, $R_2(\vec{u}_2)$, …, $R_n(\vec{u}_n)$, is called the *body* of r. The literals in the body can have input predicates, invented predicates, or output predicates, while the head of the rules must have either invented predicates or output predicates. A variable that occurs in the head must appear at least once in the body to bound the variables. The size of a rule is defined as the number of literals in its body. $|P|$, the size of a query P, is defined as the sum of the size of rules in its body.

A fragment of these queries called *union of conjunctive queries* (UCQ) is of interest in Section 4. A UCQ consists of rules where the body of the rule comprises only of input predicates. UCQ is equivalent to select-project-join queries in relational algebra [3].

2.2 Semantics of Relational Queries

The semantics of a relational query may be specified in multiple equivalent ways; see [1] for an overview. In this paper, we will formalize their semantics using rule instantiations and derivation trees. We first fix a data domain D, whose elements we will call *constants*. A *tuple*, $R(c_1, c_2, \ldots, c_k)$, consists of a k-ary relation name R with a list of constants, $c_1, \ldots, c_k$.

Given a map v from variables to the data domain D, we can *instantiate* a rule by consistently replacing its variables x with constants $v(x)$:

$$R_h(v(\vec{u}_h)) \iff R_1(v(\vec{u}_1)), R_2(v(\vec{u}_2)), \ldots, R_n(v(\vec{u}_n)).$$

Given a query P and a valuation of the input relations I, a *derivation tree* of a tuple t is a labelled rooted tree where: (a) each node of the tree is labeled by a tuple, (b) each leaf is labeled by a tuple in I; (c) the root node is labeled by t; and (d) for each internal node labeled α, there exists an instantiation $\alpha \iff \beta_1, \ldots, \beta_n$ of a rule in P such that the children of the node are respectively labelled $\beta_1, \ldots, \beta_n$. We say that a query P derives t using I if there exists a derivation tree for t.

2.3 Relational Query Synthesis Problem

Our ultimate goal is to synthesize a recursive relational query which is consistent with given input-output examples. Given (I, O^+, O^-), where I is a set of input tuples and O^+ and O^- are a partition of the output tuples as positive and negative examples respectively, a query P is said to be *consistent* with (I, O^+, O^-) if $O^+ \subseteq [\![P]\!](I)$ and $[\![P]\!](I) \cap O^- = \emptyset$.

Problem 2.1 (Query Synthesis)**.** *Given a set of input-output examples $E = (I, O^+, O^-)$ find a relational query P such that P is consistent with E.*

SYNT'22, August 11, 2022, Haifa, Israel
2022.

3 DECIDABILITY AND COMPLEXITY

We will now show that checking whether a synthesis problem instance is solvable is co-NP complete. One of the main ingredients of this proof will be the following construction:

Let the data domain $D = \{c_1, c_2, \ldots, c_n\}$, and the input tuples $I = \{R_1(\vec{c}_1), R_2(\vec{c}_2), \ldots, R_n(\vec{c}_n)\}$. Then, for $t = R(\vec{c})$, we then define the rule $r(t)$ as follows:

$$r(t) : R(\vec{v}) \text{:- } R_1(\vec{v}_1), R_2(\vec{v}_2), \ldots, R_n(\vec{v}_n).$$

where the head $R(\vec{v})$ and the body literals $R_i(\vec{v}_i)$ are obtained by by consistently replacing the constants in the output tuple $R(\vec{c})$ and input tuples $R_i(\vec{c}_i)$ with fresh variables v_c. The idea is that the body of this rule captures *all* patterns which exist among the input tuples. The rule $r(t)$ is therefore the strongest query in this data which also produces t. This gives us the following lemma:

Lemma 3.1. *Given a problem instance $E = (I, O^+, O^-)$, let $Q_{O^+} = \{r(t) : t \in O^+\}$. The problem instance admits a solution if and only if Q_O^+ is consistent with E.*

Proof. One direction of the claim is immediate: if Q_{O+} is consistent with E, then the problem admits a solution.

In the reverse direction, suppose that Q_{O+} is not consistent with E but there exists a query P consistent with E. Observe that for each $t \in O^+$, the rule $r(t)$ can produce it by picking an appropriate instantiation with which it was constructed. Hence, there exists a tuple $t^- \in O^-$ that is produced by Q_{O^+}. We will show that P also produces t^- and establish a contradiction.

Since P is consistent with E, $t \in [\![P]\!](I)$. Let τ be the derivation tree which produces t. Pick the variable valuation $\gamma : X \to D$ which causes $r(t)$ to produce the tuple t^-. Let $v : D \to X$ be the map that was used to construct $r(t)$. Apply the constant renaming map $f = \gamma \circ v : D \to D$ to every node of the derivation tree τ to produce the renamed tree $f(\tau)$. Observe that $f(\tau)$ is still a well-formed derivation tree of the query P, and that $f(t) = t^-$. It follows that the query P also produces t^- as an output tuple, contradicting our assumption that P was consistent with E. □

We now establish our main complexity result, which follows from Claims 3.3, 3.4, and 3.5.

Theorem 3.2. *Determining whether an instance of the relational query synthesis problem admits a solution is co-NP complete.*

We devote the rest of this section to the proof of this theorem.

Claim 3.3. The problem of determining whether Q_O^+ is consistent with the input-output example $E = (I, O^+, O^-)$ is in co-NP.

Proof. By construction, $O^+ \subseteq [\![Q_{O^+}]\!](I)$ and it only remains to check that $O^- \cap [\![Q_{O^+}]\!](I) = \emptyset$. A rule $r \in Q_{O^+}$ and map v from variables to constants serve as a certificate of $O^- \cap [\![Q_O^+]\!](I) \neq \emptyset$. The certificate can be verified by confirming that the tuple derived by instantiating r with v is in O^-. It follows that checking whether $Q_{I \mapsto O^+}$ is consistent with E is in co-NP. □

To show co-NP hardness, we reduce the problem of checking whether an undirected graph $G = (V, E)$ has a clique of size k to that of determining whether an instance of the synthesis problem is *unsolvable*. Without loss of generality, assume that G does not have self-loops. Consider a set of k constants $V_k = \{v_1, v_2, \ldots, v_k\}$ disjoint from V. Then, consider the instance of the synthesis problem (I, O^+, O^-), where:

$$\begin{aligned} I &= \{\texttt{edge}(u, v) \mid (u, v) \in E\} \\ &\quad \cup \{\texttt{edge}(v_i, v_j) \mid v_i, v_j \in V_k, v_i \neq v_j\}, \\ O^+ &= \{\texttt{clique}(v) \mid v \in V_k\}, \text{ and} \\ O^- &= \{\texttt{clique}(u) \mid u \in V\}. \end{aligned}$$

Claim 3.4. If G does not have a clique of size k, then the given instance is realizable.

Proof. Consider a query Q with only one rule:

$$\texttt{clique}(x_1) \text{ :- } \texttt{edge}(x_1, x_2), \ldots, \texttt{edge}(x_i, x_j), \ldots \texttt{edge}(x_k, x_{k-1}).$$

Where the premise consists of $\texttt{edge}(x_i, x_j)$ for $i \neq j$. If G does not have a clique, then we claim that $[\![P]\!](I) = O^+$. It is clear that $O^+ \subseteq [\![Q]\!](I)$. For sake of contradiction, let $\texttt{clique}(u) \in O^- \cap [\![Q]\!](I)$. Then, there is a map $v : \{x_1, \ldots, x_k\} \to V \cup V_k$ such that instantiating the rule r with v derives $\texttt{clique}(u)$ for some $u \in V$. I must contain a tuple $\texttt{edge}(v(x_i), v(x_j)) \in I$ for each $i \neq j$. By construction of I, if $\texttt{edge}(x, y) \in I$, then $x \neq y$, so each $v(x_i)$ is distinct. Also, we know that $u = v(x_1) \in V$ and $\texttt{edge}(u, v(x_i)) \in I$ for $2 \leq i \leq k$, hence, $v(x_i) \in V$. Let $u_k = v(x_k)$. We have distinct vertices $u_1, \ldots, u_k$ each in V such that there is an edge between them. Then, these vertices form a k-clique, contradicting the assumption. □

Claim 3.5. If G has a clique of size k, then the given instance is unrealizable.

Proof. Let the vertices $u_1, \ldots, u_k$ form a clique in G. Consider the map $\pi : V \cup V_k \to V$ where $\pi(u) = u$ for $u \in V$ and $\pi(v_i) = u_i$ for $v_i \in V_k$. For sake of contradiction, let P be a query consistent with the input-output example, and hence, $\texttt{clique}(v_1) \in [\![P]\!](I)$. The derivation tree for $\texttt{clique}(u_1)$ in P can be constructed by replacing each v by $\pi(v)$ in the derivation tree of $\texttt{clique}(v_1)$ in P. Hence, $u_1 \in [\![P]\!](I) \cap O^-$, contradicting the assumption that P is consistent with the input-output example. □

Lemma 3.1, and Claims 3.3, 3.4, 3.5 allow us to conclude the Relational Query Synthesis Problem is co-NP complete. Moreover, the Q_{O^+} construction synthesizes a polynomial sized relational query using the input-output examples.

4 MINIMAL QUERY SYNTHESIS PROBLEM

In this section, we address the complexity of synthesizing the smallest query from input-output examples for the case of UCQs as defined in Section 2.1.

While the query Q_{O^+}, as constructed in Lemma 3.1 solves Problem 2.1, it may overfit the input-output examples. Therefore, several synthesis tools add an optimization goal of synthesizing *minimal* queries that are consistent with the input-output examples [2, 5, 13].

For this section, we limit our analysis to union of conjunctive queries (UCQs) that form the non-recursive fragment of relational queries and use the definition of size as defined in Section 2.1. In Section 5, we discuss expanding the targeted fragment as well as changing the definition of size.

Problem 4.1 (Minimal Query Synthesis)**.** *Given input-output examples $E = (I, O^+, O^-)$, find a conjunctive query Q such that Q is consistent with E has the smallest size among all such conjunctive queries.*

The rest of this section is devoted to the complexity result:

Lemma 4.2. *Given input-output examples (I, O^+, O^-) and an integer k, determining if there exist a UCQ Q of size at most k that is consistent with (I, O^+, O^-) is in Σ_3^{P}.*

Proof. Following the proof of Claim 3.3, we can construct a NP oracle that given query Q and a tuple o, determines if $o \in [\![Q]\!](I)$. This gives us:

(1) A NP oracle $\mathcal{A}_1$ to check if $O^+ \subseteq [\![Q]\!](I)$, and
(2) A co-NP oracle $\mathcal{A}_2$ to check if $[\![P]\!](I) \cap O^- = \emptyset$.

Combining the two oracles, we have a Δ_2^P oracle $\mathcal{A}$ that checks if a given query Q is consistent with input-output examples. Consider a query Q of size at most k as a certificate. As it can be verified by oracle $\mathcal{A}$, we conclude that the decision problem is in $\mathrm{NP}^{\Delta_2^P}$, and therefore in Σ_3^{P}. □

5 CONCLUSIONS

The central concept behind this work is simple: studying the computational complexity of the synthesis task allows us to create more efficient tools and give guarantees about them. We conclude with an outline of how Theorem 3.2 impacts current tools, and then identify some directions for future work.

5.1 Completeness Guarantees

Given an instance of the synthesis problem (I, O^+, O^-), the construction in the proof of Lemma 3.1 gives a bound for the size of the desired query:

(1) There are at most $|O^+|$ rules, and
(2) The size of each rule is at most $|I|$.

Therefore, if there exists a query, there exists a query of size less than or equal to $K = |O^+ \times I|$. This bound allows us to give completeness guarantees for several existing tools.

For instance, ILP based tools such as ILASP [4] use mode declarations. The mode declarations specify the bounds on the number size of each rule, the number of variables that can be used, as well as the number of times a predicate can occur in such rules. Using these *maximal mode declarations* recovered from the construction in the proof of Lemma 3.1, allows us to give a completeness guarantee:

Theorem 5.1. *ILASP can synthesize a relational query consistent with given input-output examples using maximal mode declarations if and only if such a query exists.*

Similarly, candidate rules, metarules, and other forms of templates used by state-of-the-art tools can be further refined to give completeness results [6, 7, 9, 11].

Additionally, the bound on the size of the query gives a threshold for terminating enumerative tools such as GenSynth and Scythe [5, 14]. When all queries up to the bound K (or sketches with a corresponding bound) have been enumerated, one can abandon the search and conclude that the instance is unsolvable. We get a completeness guarantee analogous to Thereom 5.1 for enumerative tools.

While EGS has a completeness guarantee for UCQs, our result strengthens it to include all relational queries (including recursive queries, as defined in Section 2.1):

Theorem 5.2. *EGS synthesizes a relational query consistent with a given set of input-output examples if and only if such a (potentially recursive) query exists.*

5.2 Future Work

There are three directions for future work. Firstly, we described the complexity of synthesizing relational queries in a restricted fragment (positive Datalog). The completeness guarantees in this section are therefore limited, while many tools support queries with more features (including ILASP and Scythe). A direction of future work is to study the complexity of synthesizing queries with the said features (such as Answer Sets, SQL, and Prolog). A preliminary step is to incrementally study the extensions of positive Datalog to include features such as negation, aggregation, and constant comparison.

Secondly, different tools use varied language biasing mechanisms. Establishing completeness guarantees for them analogous to Theorem 5.1 requires identifying how these language biases can be recovered from the construction in Lemma 3.1 or analogous results.

And finally, it remains to determine if a stronger claim can be made for Lemma 4.2, as well as if the hardness of the problem can be established. Additionally, this result is sensitive to the definition of the size of the query. For instance, if the size of the UCQ is defined as the length of the shortest rule, the problem can be solved in Σ_2^{P}. This invites a study of all three, the computational complexity of the minimal query synthesis problem, minimal query synthesis problem for different fragments of relational queries (to support recursion and other features), and formulating an appropriate definition of the size of a query.

The results in this paper are only an initial exploration of the complexity of query synthesis problems. Given the rich variety of features for relational queries, as well as the diverse target languages for programming-by-examples in general, the study of complexity and decidability of synthesis tasks promises to be a particularly fertile topic.

Acknowledgement: The authors were supported, in part, by grants from the AFRL (FA8750-20-2-0501), ONR (N00014-18-1-2021), and the NSF (1836822). Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the Air Force Research Laboratory, the Office of Naval Research, or the National Science Foundation.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[2] Andrew Cropper, Sebastijan Dumancic, and Stephen H. Muggleton. 2020. Turning 30: New Ideas in Inductive Logic Programming. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*
[3] C.J. Date. 2003. *An Introduction to Database Systems* (8 ed.). Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Mark Law, Alessandra Russo, and Krysia Broda. 2020. The ILASP system for Inductive Learning of Answer Set Programs. *CoRR* abs/2005.00904 (2020).

[5] Jonathan Mendelson, Aaditya Naik, Mukund Raghothaman, and Mayur Naik. 2021. GenSynth: Synthesizing Datalog Programs without Language Bias. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*.

[6] Stephen Muggleton. 1995. Inverse Entailment and Progol. *New Generation Computing* 13, 3 (Dec. 1995), 245–286. https://doi.org/10.1007/BF03037227

[7] Stephen Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. 2015. Meta-interpretive Learning of Higher-order Dyadic Datalog: Predicate Invention Revisited. *Machine Learning* 100, 1 (01 July 2015), 49–73. https://doi.org/10.1007/s10994-014-5471-y

[8] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. 2011. ILP turns 20. 86, 1 (Sept. 2011), 3–23. https://doi.org/10.1007/s10994-011-5259-2

[9] Mukund Raghothaman, Jonathan Mendelson, David Zhao, Mayur Naik, and Bernhard Scholz. 2020. Provenance-guided synthesis of Datalog programs. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*.

[10] Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end Differentiable Proving. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[11] Xujie Si, Woosuk Lee, Richard Zhang, Aws Albarghouthi, Paraschos Koutris, and Mayur Naik. 2018. Syntax-guided Synthesis of Datalog Programs. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. https://doi.org/10.1145/3236024.3236034

[12] Xujie Si, Mukund Raghothaman, Kihong Heo, and Mayur Naik. 2019. Synthesizing Datalog programs using numerical relaxation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

[13] Aalok Thakkar, Aaditya Naik, Nathaniel Sands, Rajeev Alur, Mayur Naik, and Mukund Raghothaman. 2021. Example-guided synthesis of relational queries. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*.

[14] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive SQL queries from input-output examples. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*.