A Paradigm for Learning Queries on Big Data

A Paradigm for Learning Queries on Big Data

Angela Bonifati Radu Ciucanu Aurélien Lemay Sławek Staworko
University of Lille & INRIA, France
{angela.bonifati, radu.ciucanu, aurelien.lemay, slawomir.staworko}@inria.fr

ABSTRACT

Specifying a database query using a formal query language is typically a challenging task for non-expert users. In the context of big data, this problem becomes even harder as it requires the users to deal with database instances of big sizes and hence difficult to visualize. Such instances usually lack a schema to help the users specify their queries, or have an incomplete schema as they come from disparate data sources. In this paper, we propose a novel paradigm for interactive learning of queries on big data, without assuming any knowledge of the database schema. The paradigm can be applied to different database models and a class of queries adequate to the database model. In particular, in this paper we present two instantiations that validated the proposed paradigm for learning relational join queries and for learning path queries on graph databases. Finally, we discuss the challenges of employing the paradigm for further data models and for learning cross-model schema mappings.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query Languages*; I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning*

Keywords

Query inference, user interactions, learning, big data.

1. INTRODUCTION

Specifying a database query using a formal query language is typically a challenging task for non-expert users. In the context of big data, this problem becomes even harder as it requires the users to deal with database instances of big sizes and hence difficult to visualize. Such instances usually lack a schema to help the users specify their queries, or have an incomplete schema as they come from disparate data sources. Consequently, the important size of the database instances and the absence of proper metadata make unfeasible traditional query specification paradigms for non-expert users, such as query by example [40].

In this paper, we propose a novel paradigm for interactive query learning on big data, without assuming any knowledge of the database schema. This paradigm can be applied to different database models (e.g., relational, semi-structured, graph) and a class of queries adequate to the database model (e.g., join queries for the relational data model, regular path queries for the graph data model).

An important part of the paradigm is a function that identifies in the input database instance a set of small, easy to visualize fragments, from which we ask the user to label elements of interest, depending on whether or not she would like them as part of the query result. Such labeled fragments are then given as input to a query learning algorithm that returns a query that ideally expresses the information need of the user. We iterate the learning process by proposing to the user to label new fragments until her goal query is learned.

Another important aspect of the proposed paradigm is measuring the potential information that labeling a given fragment may have on the learning process: maximizing this measure has the purpose of minimizing the number of user interactions necessary to reach her goal query. Choosing the fragment that maximizes this measure may introduce significant computational and combinatorial obstacles, and to overcome them we have to investigate diligent strategies.

The main contribution of this paper is the formalization of the aforementioned paradigm (Section 2). Then, we point out that we have validated two instantiations of the paradigm for learning relational join queries [11, 12] and for learning graph queries [10]; we briefly present our results on these two application settings in Section 3 and Section 4, respectively. We discuss related work in Section 5. Finally, in Section 6 we state some future challenges related to employing the paradigm for semi-structured data (such as XML or NoSQL data stores) and for learning schema mappings between heterogeneous data models.

2. PARADIGM

Let us assume a large database instance I expressed in a given data model. Let us also suppose that we have no knowledge on whether or not the instance complies to a database schema. As an example, I can be a relational table with millions of tuples or a graph database with millions of nodes, and in either case we do not assume any knowledge of the integrity constraints. Furthermore, we do not necessarily

assume that the instance I is normalized since it often occurs in practice that users have to deal with denormalized data coming from multiple sources (e.g., obtained after some data integration scenario [20]).

If a non-expert user is willing to query such a large instance, she would not be able to formulate her query with a formal query language. Luckily, what the user is akin to do is to visualize and label fragments of this large instance depending on whether or not she would like the fragments as part of the query result. Therefore, our goal is to exploit these labeled fragments and to construct the user's goal query, which in particular satisfies the labels provided by the user. We assume that the goal query that the user has in mind belongs to a class of queries $\mathcal{Q}$ that is adequate to the model of the instance that she wants to query. In the remainder, we refer to the class of queries $\mathcal{Q}$ that contains the user's goal query as the *goal query class*. For example, in the relational case, $\mathcal{Q}$ may be the class of join queries or, in the graph case, $\mathcal{Q}$ may be the class of regular path queries.

Furthermore, for each database model, we assume a *function* that maps an instance of that model to the set of all its *fragments*. A fragment F is a small part of the instance that the user can visualize and label. For the relational model, a fragment may be a tuple or a set of tuples, depending on the actual goal relational query class. For the graph data model, a fragment may be just a node in the graph, or a small subgraph including the surroundings of the node to let the user seek the paths of interest starting in that node. The simplest possible labeling that the user can provide is a Boolean labeling (*positive* or *negative*), to indicate whether a fragment should or should not be selected by the query that she has in mind. However, our paradigm is generic enough to accommodate other kind of input, such as more complex labeling (e.g., the user may indicate whether a fragment should be selected with a certain confidence). For ease of exposition, in the rest of the paper we focus on the simplest user input using as positive and negative examples the fragments selected or unselected by the user's query, respectively.

A natural question that arises is how we choose the fragments that are proposed to the user for labeling. This question leads to describe a desirable requirement of our paradigm: we would like to minimize the amount of effort provided by the user or, in other words, we would like that the user only labels a small number of fragments in order to learn her goal query.

The fundamental steps of our paradigm are depicted in Figure 1. Before discussing each step in details, we would like to spend a few words on the following *interactive scenario* of query learning, which constitutes the core of our paradigm. Let us consider an instance and an empty set of examples S. We may also refer to a set of examples as a *sample*. We say that a query q is *consistent with* S if q selects all positive examples and none of the negative ones. Initially, since we have an empty sample S, all queries in $\mathcal{Q}$ are consistent with S. Then, we reiterate the following process: we choose a fragment F, the user labels F, and we update the set of queries consistent with S. We stop the interactions when a sufficient knowledge of the goal query has been accumulated i.e., there exists exactly one query consistent with the examples that ideally is the user's goal query. Since we assume that the user labels the given fragments consistently with some query q that she has in mind, q always belongs to the current set of consistent queries. Moreover, since we

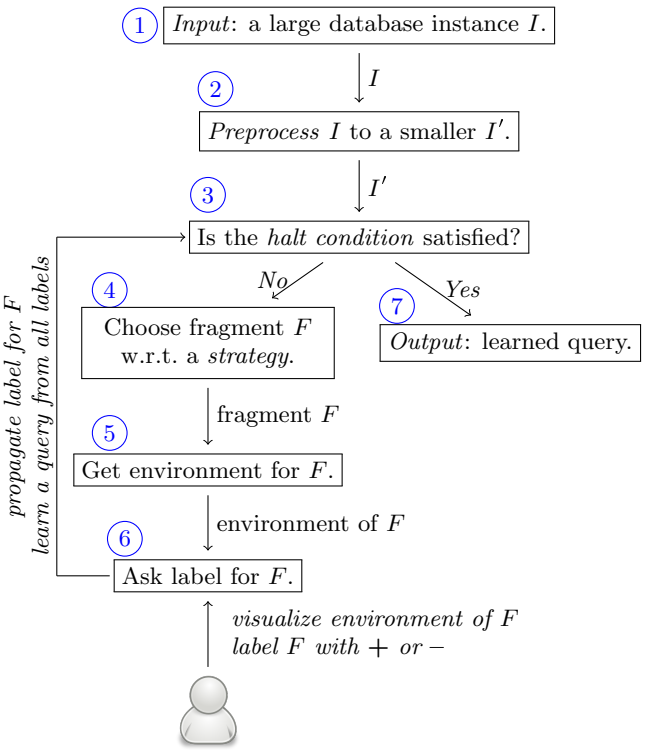

Figure 1: Workflow of the paradigm.

want to get quickly from an initial empty sample to a sample S s.t. only the goal query is consistent with it, we have to choose at each iteration the fragment F that maximizes the potential impact i.e., when the user labels F, we are able to eliminate a maximal number of candidate queries from the set of all queries consistent with S. To this purpose, it is important to ask the user to label only fragments that contribute to the learning process. To be able to define such fragments, we formalize first the notion of *uninformative fragments*. Given a sample S, we say that a fragment is *uninformative* if labeling it explicitly as a positive or a negative example does not eliminate any query consistent with S. Then, a fragment is *informative* if it has not been labeled by the user nor it is uninformative. Computing the set of informative fragments actually corresponds to *propagating* every label given by the user to the fragments that are still unlabeled and removing those that become uninformative because of that label.

What we have described above is the interactive scenario, which is the core of our paradigm. This helps us detail the steps of the paradigm, depicted in Figure 1, in the rest of this section.

(1) (2) The paradigm takes as *input* a large database instance I. Unfortunately, working on the initial instance I is unfeasible due to the fact that all the instance would have to be inspected to seek relevant fragments that we propose to the user to label. Therefore, a first important step of our paradigm is the *preprocessing*, which produces an instance I' that is considerably smaller than the initial I. We point out that I' should ideally contain sufficiently diverse fragments to permit learning all queries from the targeted class

$\mathcal{Q}$. As such, the size of I' depends more on the class of queries $\mathcal{Q}$ than on the initial instance I. A simple way to obtain I' from I is to remove all redundant fragments. For example, if we want to assist a user to learn a relational join query, the preprocessing could consist of simply removing the tuples that are equivalent in the sense that they are selected exactly by the same set of joins. Similarly, if we want to learn a regular path query on graphs, the preprocessing could consist of taking a subgraph of the initial large graph on which we can find the same paths as in the entire graph.

(3) The interactions with the user continue until a *halt condition* is satisfied. A natural halt condition is to stop the interactions when there is exactly one consistent query with the current sample. In practice, we can imagine weaker conditions e.g., the user may stop the process earlier if she is satisfied by some candidate query proposed at some intermediary stage during the interactions.

(4) The fragments shown to the user are chosen according to a *strategy* i.e., a function that takes as input an instance I' and a sample S, and returns a fragment from I'. Since we want to minimize the amount of examples needed to learn the user's goal query, an intelligent strategy should propose to the user only informative fragments. We point out that while it is possible to design an optimal strategy (i.e., that is guaranteed to propose to the user a minimal number of fragments), such a strategy is usually based on a minimax algorithm, thus being exponential and unfortunately unfeasible in practice. This motivates us to investigate *practical strategies* (i.e., that efficiently compute the next fragment to propose to the user) also because we do not want make the user wait too long between two consecutive interactions. This approach leads to defining the *entropy* of a fragment, which intuitively is a measure of the quantity of information that labeling that fragment brings to the learning process. The computation of the entropy of a fragment is related to the actual data model and to the goal query class. We have already defined such measures for learning relational joins [11] and for learning path queries on graphs [10]. In any case, an intelligent strategy proposes to the user a fragment that maximizes the entropy.

(5) A fragment by itself does not always carry enough information to allow the user understand whether the fragment is part of the query result or not. Therefore, it may happen that we have to enhance the information of a fragment by zooming out on the *environment* of a such fragment before actually showing it to the user. This step is rather trivial in the case of relational joins, where the fragment is a tuple and the user has to visualize it entirely before deciding if this tuple should be selected. However, there may be the case that the instance consists of a denormalized table having a large number of attributes and consequently we may need to select the most relevant ones to not overwhelm the user. Moreover, the step of constructing the environment of a fragment may become cumbersome in the case of queries on less structured databases (such as trees or graphs). Hence, a challenge of the algorithm is to compute a small environment of a node that is easy to visualize by the user and rich enough to permit labeling.

(6) (7) The user visualizes the environment of a given fragment F and labels F w.r.t. the goal query that she has in mind. In Figure 1, we have depicted the simplest fragment labeling i.e., as positive "+" or negative "–" examples. Then, we propagate the label given by the user for F in the rest of the instance to prune the fragments that become uninformative. Moreover, we run a *learning algorithm* (i.e., a function that takes as input an instance and a sample, and outputs a query consistent with the sample) to propose the "best" query that is consistent with all labels provided until this point. When the halt condition is satisfied, we return the latest learned query to the user. In particular, the halt condition may take into account such an intermediary learned query q e.g., when the user is satisfied by the output of q on the instance and wants to stop the interactions.

3. LEARNING RELATIONAL QUERIES

In this section, we present an instantiation that validated the proposed paradigm in the context of *relational databases*, in particular for *learning join queries*. In our work [11, 12], we have studied the following setting: the input *instance* consists of a collection of relations (that can be viewed as a unique denormalized table corresponding to their Cartesian product), the *fragments* are individual tuples from that instance, and the *goal query class* consists of the set of all *join predicates* that can be formulated over it.

For example, assume a scenario where a user working for a travel agency wants to build a list of flight&hotel packages. The user is not acquainted with querying languages and can only access the information on flights and hotels in a denormalized table, such as the one in Figure 2. We

	From	*To*	*Airline*	*City*	*Discount*	
	Paris	Lille	AF	NYC	AA	(1)
	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)
+	Lille	NYC	AA	NYC	AA	(4)
	Lille	NYC	AA	Paris	None	(5)
	Lille	NYC	AA	Lille	AF	(6)
	NYC	Paris	AA	NYC	AA	(7)
–	NYC	Paris	AA	Paris	None	(8)
	NYC	Paris	AA	Lille	AF	(9)
	Paris	NYC	AF	NYC	AA	(10)
	Paris	NYC	AF	Paris	None	(11)
	Paris	NYC	AF	Lille	AF	(12)

Figure 2: A set of tuples.

assume no knowledge of the schema and of the provenance of the data. Assuming that the user has labeled the tuple (3) as a positive example, note that tuple (4) becomes uninformative. Moreover, if we assume that the user has labeled (8) as a negative example, the only consistent join predicate is $\{(To, City), (Airline, Discount)\}$, defining a query that intuitively selects packages consisting of a flight and a stay in a hotel combined in a way allowing a discount.

We show next a simple but effective way to *preprocess* such an instance for the purpose of learning join queries on it. Given a tuple t, by $T(t)$ we denote the most specific join predicate selecting t. The set of all such predicates together with the $\subseteq$ relation form a *lattice of join predicates* having as bottom-most element the most general join predicate $\varnothing$ and as top-most element the most specific join predicate Ω (that possibly selects no tuple on the instance). In Figure 3 we present the lattice obtained for the instance in Figure 2.

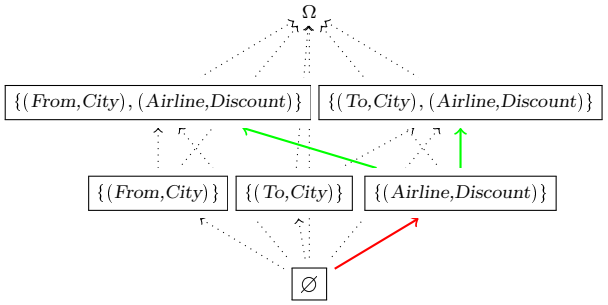

Figure 3: Lattice of join predicates for Figure 2.

From	*To*	*Airline*	*City*	*Discount*	$T(t)$
Paris	Lille	AF	NYC	AA	$\varnothing$
Paris	Lille	AF	Paris	None	{(*From*,*City*)}
Paris	Lille	AF	Lille	AF	{(*To*,*City*), (*Airline*,*Discount*)}
NYC	Paris	AA	NYC	AA	{(*From*,*City*), (*Airline*,*Discount*)}
NYC	Paris	AA	Paris	None	{(*To*,*City*)}
Paris	NYC	AF	Lille	AF	{(*Airline*,*Discount*)}

Figure 4: Preprocessed instance for Figure 2.

Then, for each predicate θ that selects at least one tuple in the instance in Figure 2 (i.e., those represented in a box in Figure 3), we take one tuple t from the instance s.t. $T(t) = \theta$ and we add t in the preprocessed instance. For the instance in Figure 2, we obtain the preprocessed instance in Figure 4 (where we present on the last column the corresponding $T(t)$ for each tuple). Notice that with this simple preprocessing procedure we have eliminated half of the tuples from the initial set, without altering the set of queries that can be learned on this instance.

In [11], we have employed this preprocessing technique and shown that it scales for instances of millions of tuples. Moreover, we have developed a set of efficient strategies of proposing tuples to the user and we have characterized in which cases different classes of strategies are expected to perform better than other ones. Here, we only give the intuition behind these strategies and we point out that they are based on reasoning on the lattice of join predicates. As we have already shown, there is a correspondence between the tuples in the preprocessed instance and the elements of the lattice. Thus, eliminating uninformative tuples can be seen as a label propagation in the lattice. For example, assume an empty sample and that we ask the user to label the tuple corresponding to the join predicate {(*Airline*,*Discount*)}. If the user labels it as a positive example, we are able to eliminate the tuples corresponding to predicates above it in the lattice (i.e., {(*From*,*City*), (*Airline*,*Discount*)} and {(*To*,*City*), (*Airline*,*Discount*)}); conversely, if the user labels it as a negative example, we eliminate the tuples corresponding to tuples below it in the lattice (i.e., $\varnothing$). Intuitively, the question "Which is the next tuple to present to the user?" becomes "Labeling which tuple allows us to eliminate as many elements of the lattice as possible?" Finally, we point out that we have implemented the proposed strategies in Jim (Join Inference Machine) [12], a system that can learn arbitrary n-ary join predicates, spanning from relational tables to sets of tagged images.

4. LEARNING GRAPH QUERIES

In this section, we present an instantiation of our paradigm in the context of *graph databases*. In our work [10], we have studied the following setting: the input *instance* consists of a graph database, the *fragments* are nodes of the graph, and we have focused on a *goal query class* for graphs that we detail later in this section.

A graph database instance is essentially a directed, edge-labeled graph [7, 36]. As an example, take in Figure 5 a geographical graph database having as nodes cities (C_1 to C_6), hospitals (H_1 and H_2), and schools (S_1 and S_2). The edges represent transportation facilities between cities (using labels *train* and *bus*) or other kind of facilities (using labels *hospital* and *school*). For instance, the edge "(C_2, *bus*, C_3)" means that one can travel by bus between C_2 and C_3, while the edge "(C_4, *hospital*, H_1)" means that the hospital H_1 is situated in the city C_4.

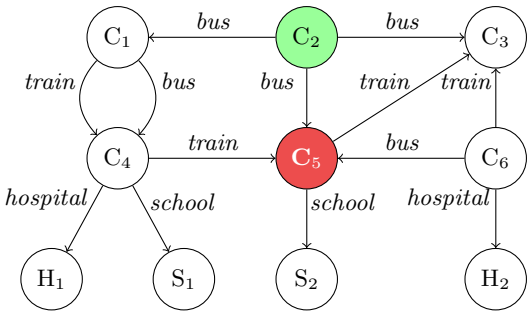

Figure 5: A geographical graph database.

We have focused on the class of *monadic regular path queries* that intuitively select nodes having at least one node in the language of a given regular expression. For example, the following query selects the cities from which one can reach hospitals in the graph database from Figure 5:

$$q = (train + bus)^* \cdot hospital.$$

This query selects the nodes C_1, C_2, C_4, and C_6 because there exist the following *paths* in the graph:

$C_1 \xrightarrow{train} C_4 \xrightarrow{hospital} H_1$, $\quad C_4 \xrightarrow{hospital} H_1$,

$C_2 \xrightarrow{bus} C_1 \xrightarrow{train} C_4 \xrightarrow{hospital} H_1$, $\quad C_6 \xrightarrow{hospital} H_2$.

Although the *fragments* that the user labels are individual *nodes*, for visualization purposes we also show to the user the immediate environment of the candidate nodes to let her identify the paths starting from those nodes. This environment intuitively consists of all nodes reachable from the proposed node, with paths of a certain length (in [10], we have characterized theoretically and empirically this length). Assuming that the user visualizes the graph in Figure 5 and that she labels the node C_2 as a positive example and the node C5 as a negative example, notice that the above q is clearly consistent with these labels. Furthermore, all nodes without any outgoing edge (i.e., H_1, H_2, S_1, S_2, C_3) are uninformative with $-$, and C_1, C_4, C_6 are informative nodes.

We have shown in [10] that applying our paradigm for learning this kind of queries is much more difficult than for relational joins, for the following two reasons: (i) given a set of positive and negative node examples, there may exist an infinite number of queries consistent with the labels, and

(ii) deciding whether a consistent query does even exist is an intractable problem. Therefore, since the problem of finding a consistent query implies high computational costs, we may not propose a candidate "best" query after each new label provided by the user, but we may go directly to asking her to label a new fragment. We have precisely characterized in [10] the conditions that a given set of examples should satisfy to permit learning of the goal query in polynomial time. As for the preprocessing of large graph databases, we point out that this problem reduces to large graph sampling [26] i.e., computing a representative subgraph of the initial large graph that permits learning of any monadic regular path query over the set of labels used in the initial graph.

5. RELATED WORK

First, we discuss related work w.r.t. the general aspect of our paradigm and then the related work w.r.t. learning relational and graph queries, respectively.

General aspect of our paradigm. For the validation of our paradigm [10, 11, 12], we have considered the simplest possible user input by requiring the user only to answer *Yes* or *No* to simple fragment labeling. This type of interactions is inspired by the well-known framework of *learning with membership queries* proposed by Angluin [6]. Nonetheless, our paradigm is generic enough to allow other kind of user feedback. For example, existing researches on exploratory querying big data collections (e.g., [29]) can be seen as instantiations of our paradigm with an expert user that provides more complex input. Additionally, our notion of entropy is related to a similar notion used in [29].

Our notion of uninformative fragments is inspired by possible world semantics and certain answers [22]. In particular, a similar notion has been already employed in the context of XML querying for non-expert users [17]. The idea of measuring the entropy (i.e., the information gain of labeling a given fragment) is strongly connected to the notion of active learning [30]. Active learning have been already used in the context of learning to repair constraint violations from user feedback [37] and in the context of keyword search-based data integration [32, 38].

Moreover, the idea of our paradigm to minimize the number of user interactions is related to *crowdsourcing* applications, where minimizing the number of interactions entails lower financial costs. A difference is that a crowd scenario considers multiple users providing the answers. However, one can assume in a black-box the steps of drawing the crowd users and of corroborating their answers (in the spirit of [5]) and then employ our paradigm for efficiently learning a goal query while minimizing the number of crowd interactions.

Learning relational queries. The instantiation of our paradigm for learning relational queries [11, 12] follows a very recent line of research [39, 34, 18] from which we differ in two ways: (i) we assume no knowledge of the database schema, and (ii) we do not have an initial query output to start with and we discover it from user interactions. Another work strongly related to ours is [1, 2], which focuses on *learning quantified Boolean queries* and also uses the framework of learning with membership queries [6]. The goal of their system is somewhat different from our paradigm, in that their goal is to disambiguate a natural language specification of the query, whereas we focus on raw data to guess the "unknown" query that the user has in mind. Additionally, our work on learning relational joins is connected to join processing using the crowd. However, crowdsourced joins have been mainly defined in terms of entity resolution, where joining two datasets means finding all pairs of tuples that refer to the same entity [27, 35]. Conversely, we have handled arbitrary n-ary join predicates in our instantiation, thus targeting a quite different and more intricate goal for the crowd i.e., learning such join predicates from a set of positive and negative labels.

Learning graph queries. The learnability definition that we have used in [10] is inspired by *grammatical inference* [19] i.e., the branch of machine learning that aims at constructing a formal grammar by generalizing a set of examples. Grammatical inference techniques have been recently employed in the context of XML for learning queries [13, 25, 31], schemas [8, 9, 16], and transformations [23, 24]. More precisely, in [10] we have used a variant of the classical grammatical inference framework (i.e., language identification in the limit with polynomial time and data [21]) that takes into account the intractability of deciding the consistency of a set of examples (in the spirit of [23]).

6. CHALLENGES AND PERSPECTIVES

We have proposed a paradigm for learning queries on large database instances, for any database model and an adequate class of queries. We have shown two instantiations that validated our approach: for learning relational join queries [11, 12] and for learning monadic regular path queries for graphs [10]. We end this paper with two challenges of applying our paradigm in different settings that we would like to investigate in the future.

Learning queries for semi-structured databases. Learning XML queries have been studied already either without user interactions [31] or in an interactive setting without having as goal to minimize the number of examples needed for learning the goal query [13, 25]. We would like to investigate in the future the problem of learning queries for semi-structured databases in an interactive scenario in the spirit of our paradigm. For this purpose, we would have to base ourselves on the algorithms from [13, 25, 31] for learning a query from a set of examples and on the algorithms from [17] for pruning the uninformative fragments. Additionally, we would like to apply our paradigm for learning queries for NoSQL document data stores [14] that are structured as trees, similarly to the XML databases.

Learning cross-model schema mappings. A more challenging task would be to apply our interactive paradigm to learning cross-model schema mappings. A schema mapping is essentially a logical assertion between two queries, one on a source database and the other on the target database. Designing a schema mapping via data examples has been investigated in [3, 4, 28], but using more expert users than in our paradigm. Moreover, the problem of learning schema mappings from simple user interactions has been studied in [33], but only in the relation-to-relational case. Interactive query learning on heterogeneous data models is a fundamental step towards interactive learning of cross-model

schema mappings [15]. We believe that designing efficient algorithms for mapping big data instances to any other model that is easier to manipulate by the non-expert user (e.g., a common schema that the user feels confident to query) can bring the most value of the big data to the users. Hence, our immediate objective is to thoroughly investigate the application of the proposed interactive paradigm to efficiently learning queries on different data models before proposing interactive learning of cross-model schema mappings.

7. REFERENCES

[1] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *PODS*, pages 49–60, 2013.

[2] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Playful query specification with DataPlay. *PVLDB*, 5(12):1938–1941, 2012.

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

[4] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. EIRENE: Interactive design and refinement of schema mappings via data examples. *PVLDB*, 4(12):1414–1417, 2011.

[5] A. Amarilli, Y. Amsterdamer, and T. Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, pages 15–25, 2014.

[6] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[7] P. Barceló. Querying graph databases. In *PODS*, pages 175–188, 2013.

[8] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *TWEB*, 4(4), 2010.

[9] I. Boneva, R. Ciucanu, and S. Staworko. Simple schemas for unordered XML. In *WebDB*, pages 13–18, 2013.

[10] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases, 2014. Under submission.

[11] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive inference of join queries. In *EDBT*, pages 451–462, 2014.

[12] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive join query inference with JIM. *PVLDB*, 7(13), 2014.

[13] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. *Machine Learning*, 66(1):33–67, 2007.

[14] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2010.

[15] R. Ciucanu. Learning queries for relational, semi-structured, and graph databases. In *SIGMOD/ PODS Ph.D. Symposium*, pages 19–24, 2013.

[16] R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In *DBPL*, 2013.

[17] S. Cohen and Y. Weiss. Certain and possible XPath answers. In *ICDT*, pages 237–248, 2013.

[18] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *ICDT*, pages 89–103, 2010.

[19] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[20] X. L. Dong and D. Srivastava. Big data integration. *PVLDB*, 6(11):1188–1189, 2013.

[21] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[22] T. Imielinski and W. Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[23] G. Laurence, A. Lemay, J. Niehren, S. Staworko, and M. Tommasi. Learning sequential tree-to-word transducers. In *LATA*, pages 490–502, 2014.

[24] A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for top-down XML transformations. In *PODS*, pages 285–296, 2010.

[25] A. Lemay, J. Niehren, and R. Gilleron. Learning n-ary node selecting tree transducers from completely annotated examples. In *ICGI*, pages 253–267, 2006.

[26] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD*, pages 631–636, 2006.

[27] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[28] L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In *SIGMOD Conference*, pages 73–84, 2012.

[29] T. Sellam and M. L. Kersten. Meet Charles, big data query advisor. In *CIDR*, 2013.

[30] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[31] S. Staworko and P. Wieczorek. Learning twig and path queries. In *ICDT*, pages 140–154, 2012.

[32] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. *PVLDB*, 1(1):785–796, 2008.

[33] B. ten Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28, 2013.

[34] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *SIGMOD Conference*, pages 535–548, 2009.

[35] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD Conference*, pages 229–240, 2013.

[36] P. T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.

[37] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

[38] Z. Yan, N. Zheng, Z. G. Ives, P. P. Talukdar, and C. Yu. Actively soliciting feedback for query answers in keyword search-based data integration. *PVLDB*, 6(3):205–216, 2013.

[39] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *SIGMOD Conference*, pages 809–820, 2013.

[40] M. M. Zloof. Query by example. In *AFIPS National Computer Conference*, pages 431–438, 1975.