Interactive Join Query Inference with JIM

Angela Bonifati, Radu Ciucanu, Slawomir Staworko

Interactive Join Query Inference with JIM

Angela Bonifati
University of Lille & INRIA
angela.bonifati@inria.fr

Radu Ciucanu
University of Lille & INRIA
radu.ciucanu@inria.fr

Sławek Staworko
University of Lille & INRIA
slawomir.staworko@inria.fr

ABSTRACT

Specifying join predicates may become a cumbersome task in many situations e.g., when the relations to be joined come from disparate data sources, when the values of the attributes carry little or no knowledge of metadata, or simply when the user is unfamiliar with querying formalisms. Such task is recurrent in many traditional data management applications, such as data integration, constraint inference, and database denormalization, but it is also becoming pivotal in novel crowdsourcing applications. We present JIM (Join Inference Machine), a system for interactive join specification tasks, where the user infers an n-ary join predicate by selecting tuples that are part of the join result via Boolean membership queries. The user can label tuples as positive or negative, while the system allows to identify and gray out the uninformative tuples i.e., those that do not add any information to the final learning goal. The tool also guides the user to reach her join inference goal with a minimal number of interactions.

1. INTRODUCTION

Query specification and, in particular, join specification may become cumbersome tasks for non-expert users if they are unfamiliar with language formalisms and thus unable to manually write a join predicate. Nevertheless, join specification may become feasible for non-expert users whenever they can easily access data and metadata altogether. This happens in traditional query specification paradigms, such as query-by-example [6], that are typically centered around a single database. When it comes to consider raw data coming from different data sources, such paradigms are not applicable any longer. The reason is twofold: (i) such data may not carry pertinent metadata to be able to specify a join predicate and (ii) value-based matching of tuples is unfeasible in most cases, due to a massive number of tuples.

We present JIM (Join Inference Machine), a system that assists unfamiliar users to specify their join queries via simple tuple labeling. More precisely, the user is interactively presented with candidate tuples and is asked to label them as positive or negative depending on whether or not she would like the tuples as part of the join result. JIM is able to infer arbitrary n-ary join queries via a minimal number of user interactions and without assuming any prior knowledge of the integrity constraints between the involved relations. We also point out that JIM handles a varying number of involved relations.

In [3], we have addressed the theoretical challenges of such a scenario, proposed several strategies of presenting tuples to the user, and shown their efficiency and scalability on benchmark and synthetic datasets. We observe that the user providing the examples in the experiments from [3] is in fact a program that labels tuples w.r.t. a goal join query. As a natural extension, we are interested in applying our algorithms to realistic scenarios, where human users provide positive and negative examples for join inference. The goal of this demo is thus to allow real users to interact with JIM to infer, via a minimal number of interactions, different join queries that they could have in mind.

Since our goal is to minimize the number of interactions with the user, JIM is of interest for novel database applications, such as *joining datasets using crowdsourcing*, where minimizing the number of interactions entails lower financial costs. 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 [4, 5]. Conversely, JIM can handle arbitrary n-ary join predicates, thus targeting a quite different and more intricate goal for the crowd i.e., inferring such join predicates from a set of positive and negative labels.

As a further difference, the existing systems that allow join processing with the crowd [4, 5] do not take into account the labels already given by the user to adjust the order of presenting new tuples for labeling. On the other hand, JIM continuously interleaves the user's feedback and the inference process. Indeed, after each interaction JIM prunes the *uninformative tuples* (i.e., that do not contribute any new information about the goal query) and asks the user to label the *most informative tuple* according to a suitable *strategy*.

Moreover, JIM is also of interest for applications of *schema mapping inference*, assuming a less expert user than existing systems allowing interactive schema mapping specification via data examples [1]. Indeed, in our case the annotations correspond to simple membership queries [2] to be answered even by a user who is not familiar with schema mappings and our join queries can be eventually seen as simple GAV mappings.

Proceedings of the VLDB Endowment, Vol. 7, No. 13

The rest of the paper is organized as follows. In Section 2 we present some of the key ingredients of our system via a motivating example, while in Section 3 we describe our demonstration scenario. Due to space restrictions, in this paper we provide only a glimpse of the techniques employed by Jim. However, we refer to our full research paper [3] for more algorithmic details and also for more elements of related work.

2. SYSTEM OVERVIEW

In this section we present a brief overview of Jim. For this purpose we first introduce a *motivating example*. Then, we describe the core of Jim i.e., the *interactive scenario* for join query inference.

Motivating example

Consider 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 access the information on flights and hotels in a denormalized table as in Figure 1.

	Flight			*Hotel*		
	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 1: A set of tuples.

We assume no knowledge of the schema and of the provenance of the data. Due to this assumption, several queries can be formulated that correspond to different ways of pairing a flight and a hotel. For the sake of simplicity and clarity of the illustration, we focus on two of them in the remainder: one that selects packages consisting of a flight and a stay in a hotel and another one that additionally ensures that the package is combined in a way allowing a discount. These two queries roughly correspond to the following equijoin predicates:

(Q_1) $To = City$,

(Q_2) $To = City \wedge Airline = Discount$.

Since the user is unable to formally specify such queries with a query language, she starts to look at the tuples and indicates whether or not a given tuple is of interest to her. We view this as labeling with + and − the tuples from Figure 1. For instance, suppose the user wants the flight from Paris to Lille operated by Air France (AF) and the hotel in Lille. This corresponds to labeling by + the tuple (3) in Figure 1.

Observe that both queries Q_1 and Q_2 are consistent with this labeling i.e., both queries select the tuple (3). Naturally, the objective is to use the labeling of further tuples to identify the goal query i.e., the query that the user has in mind. Not all the tuples can however serve this purpose. For instance, if the user labels next the tuple (4) with +, both queries remain consistent. Intuitively, the labeling of the tuple (4) does not contribute any new information about the goal query and is therefore *uninformative*. Since the input tables may be big, it may be unfeasible for the user to label every tuple in the instance.

In this context, our next goal is to minimize the number of tuples that the user needs to label in order to infer her goal query. Consequently, we want to measure the quantity of information that labeling a tuple could bring to the inference process and present to the user only tuples that maximize this measure. In particular, since uninformative tuples do not contribute any new information, they are not presented to the user.

In the example of the flight&hotel packages, a tuple whose labeling can distinguish between Q_1 and Q_2 is, for instance, the tuple (8) because Q_1 selects it and Q_2 does not. If the user labels the tuple (8) with −, then the query Q_2 is returned; otherwise Q_1 is returned. We also point out that the use of only *positive* examples, tuples labeled with +, is not sufficient to identify all possible queries. As an example, query Q_2 is contained in Q_1, and therefore, Q_1 satisfies all positive examples that Q_2 does. Consequently, the use of *negative* examples, tuples with label −, is necessary to distinguish between these two.

Interactive scenario

Even though our demonstration scenario consists of several types of interactions with the user, in this section we concentrate exclusively on the core of Jim, the *interactive scenario* for join query inference, that is depicted in Figure 2.

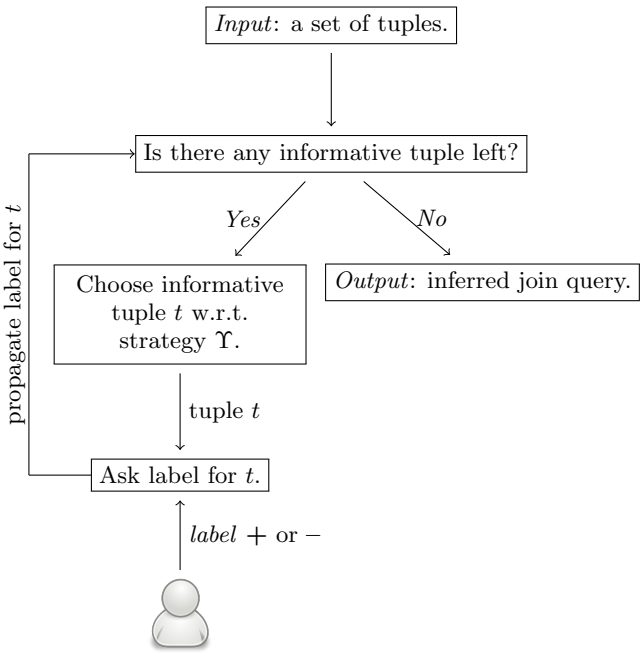

Figure 2: Interactive scenario.

During the interactive scenario, we present the user with a tuple and she indicates whether the tuple is selected or

1. Labeling all tuples.

2. Interactively graying out uninformative tuples.

3. Proposing top-k informative tuples.

4. Proposing the most informative tuple.

Figure 3: Four types of interactions with the user.

not by the join predicate that she has in mind by labeling the tuple as a positive or negative example. This process is repeated until a sufficient knowledge of the goal join predicate has been accumulated i.e., there exists a unique (up to instance-equivalence [3]) join predicate consistent with the user's labels. For instance, for the tuples in Figure 1, assuming that (3) is a positive example, and (7) and (8) are negative examples, there is only one consistent join predicate (i.e., the above Q_2). This scenario is inspired by the well-known framework of *learning with membership queries* [2].

Since the instance may be of big size, we do not want to ask the user to label all tuples, but only a small part of them. The goal of Jim is to minimize the number of interactions with the user, hence an inherent problem that it deals with is in which order to present the tuples to the user. A first remark is that we should not present at all to the user the tuples that are *uninformative* i.e., that do not contribute to the inference process. For example, given the tuples in Figure 1 and assuming that the user has labeled the tuple (3) as a positive example, note that the tuple (4) is uninformative.

Consequently, we want to ask the user to label only *informative* tuples. When the user labels an informative tuple, Jim propagates this label and prunes the tuples that become uninformative. For example, assume that Jim asked the user to label the tuple (12). If the user labels it as a positive example, we are able to prune the tuples that are selected by the most specific predicate selecting it: (3), (4), (7). Conversely, if the user labels tuple (12) as a negative example, we are able to prune the tuples that are selected by more general predicates than those selecting it: (1), (5), (9). Intuitively, the question "Which is the next tuple to present to the user?" becomes "Labeling which tuple allows us to prune as many tuples as possible?"

Next, we briefly introduce the notion of strategy for interactively presenting tuples to the user. Formally, a *strategy* Υ is a function that, given a set of tuples and some labels, returns an informative tuple. As we have pointed out in [3], there exists a algorithm that computes the *optimal strategy* of showing tuples to the user, but it requires exponential time, which unfortunately renders it unusable in practice. As a consequence, we have proposed a number of time-efficient strategies that attempt to minimize the number of interactions with the user.

More precisely, the strategies proposed in [3] and implemented in Jim are essentially classified in two categories: *local* and *lookahead*, and for comparison we have also introduced the *random* strategy which chooses randomly an informative tuple. They key difference between local and lookahead strategies is that the local ones are rather simple and based on some fixed orders, while the lookahead ones take into account the quantity of information that labeling an informative tuple could bring to the inference process, by using a generalized notion of *entropy*. The construction of all these strategies is quite technical and we refer to [3] for more details and for comprehensive experiments showing precisely in which cases different classes of strategies are expected to perform better than the other ones.

3. DEMONSTRATION SCENARIO

The demonstration scenario consists of three parts. First, we want to make the attendee aware of the fact that by using an interactive approach, Jim saves a lot of effort in specifying join queries. Next, we want to give to the attendee the insight behind how the choice of a strategy influences the behavior of Jim. Finally, the attendee will be able to use Jim to specify joins not only between relational tables, but also between sets of tagged pictures.

Why using a strategy?

To illustrate why it is important to employ an "intelligent" strategy of proposing tuples to the user, we progressively present four types of interactions, as shown in Figure 3:

1. We let the attendee choose the tuples that she wants to label as positive and negative examples, in any order she prefers.

2. We still let the attendee label tuples in any order, but after each given label we interactively gray out the tuples that become uninformative.

3. We compute the top-k informative tuples and we ask the attendee to label only them.

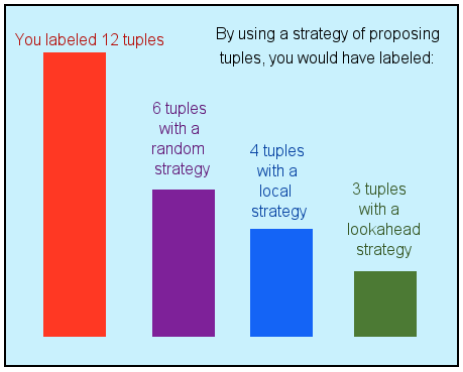

Figure 4: Showing the benefit of using a strategy.

4. We apply the interactive inference process described in the previous section i.e., we interactively propose the most informative tuple until we infer the goal query.

For each of (1), (2), and (3), after inferring the join query, we also show graphically to the attendee (as in Figure 4) how many interactions she would have done if she had used a strategy of proposing informative tuples to her. Moreover, we always show in our interface basic statistics about the progress of learning: the total number (and the relative percentage) of tuples that have been explicitly labeled by the user or deemed as uninformative, etc.

Although the core of Jim is made of interactions of type (4), we consider that it is also important to show the other types in the demonstration and the reasons are twofold. First, we want to show that Jim requires very few labeled tuples to infer the goal query. Moreover, we want to show that the amount of user effort in specifying her join query is minimal, in the sense that she only has to answer "Yes/No" to a tuple proposed by Jim. This feature clearly saves the user's time compared to the case when the user has to look at all the tuples and decide which one to label.

Comparing different strategies

In this part of the demonstration we focus only on the interactive inference process described in the previous section and on interactions of type (4), also depicted in the fourth screenshot of Figure 3. The goal here is to make the attendee understand the cases when local or lookahead strategies are better fitted. More concretely, we let the attendee infer with our system more or less complex join queries on different instances and after each inference we show how many interactions she did compared to the number of interactions that she would have done using the other strategies. The graphical presentation is done in the same spirit as in Figure 4 for the previous part. The attendee must observe that for more complex instances and join queries a lookahead strategy performs better than a local one while for simpler instances and queries a local strategy is better.

Joining sets of pictures

In this last part of the demonstration we show that our system is able to infer joins not only between relational tables, but also between different types of tagged media. An example of preloaded database consists of the cards used in

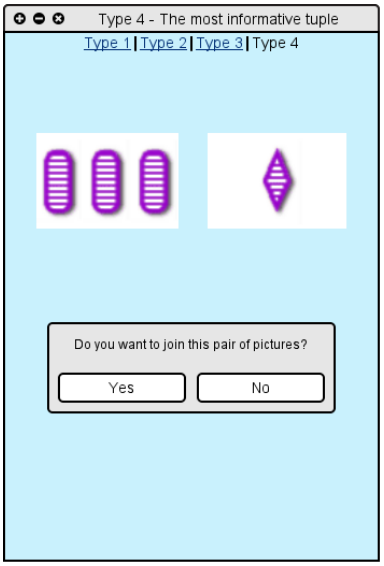

Figure 5: Interactively joining sets of pictures.

the game Set[1], which vary in four features: number (one, two, or three), symbol (diamond, squiggle, oval), shading (solid, striped, or open), and color (red, green, or purple). As already explained for the interactive inference process, we repeatedly show the most informative pair of pictures to the attendee that she labels as positive or negative until we infer her join query (as in Figure 5). In this part of the scenario, the attendee can train the system to infer a n-ary join predicate of the form: "*select the pairs of pictures having the same color and the same shading.*" We point out that this feature of Jim of joining sets of pictures using a minimal number of simple interactions is of interest for crowdsourcing applications where this kind of task is quite common and crucial. Moreover, note that Jim handles arbitrary n-ary join queries (such as the binary one mentioned above), a feature that goes beyond the usual definition of crowdsourced joins.

4. REFERENCES

[1] 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.

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

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

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

[5] 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.

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

[1] `http://www.setgame.com/set`