selP: Selective Tracking and Presentation of Data Provenance

Daniel Deutch, Amir Gilad, Yuval Moskovitch
Computer Science Department, Tel Aviv University
{danielde,amirgilad,moskovitch1}@post.tau.ac.il

***Abstract*—Highly expressive declarative languages, such as *Datalog*, are now commonly used to model the operational logic of data-intensive applications. The typical complexity of such Datalog programs, and the large volume of data that they process, call for the tracking and presentation of *data provenance*. Provenance information is crucial for explaining and justifying the Datalog program results. However, the size of full provenance information is in many cases too large (and its concise representations are too complex) to allow its presentation to the user. To this end, we propose a demonstration of selP, a system that allows the selective presentation of provenance, based on user-specified top-k queries. We will demonstrate the usefulness of selP using a real-life program and data, in the context of Information Extraction.**

I. Introduction

Many real-life applications rely on an underlying database in their operation. In different domains, such as declarative networking [1], social networks [2], and information extraction [3], it has recently been proposed to use *Datalog* for the modeling of such applications. The Datalog programs employed in these contexts are typically quite complex, calling for *explanations* of the way results were obtained.

Consider for example AMIE [3], a system for mining logical rules from Knowledge Bases (KBs), based on observed correlations in the data. After being mined, these rules are then treated as a Datalog program [1] and evaluated with respect to a KB of facts (e.g. YAGO) that were directly mined form sources such as Wikipedia. This allows to address incompleteness of Knowledge Bases, gradually deriving additional new facts and introducing them to the KB. However, since the rules are automatically mined, there is an inherent uncertainty with respect to their validity. Indeed, many rules mined in such a way are not universally valid, but are nevertheless very useful (and used in practice), since they contribute to a higher recall of facts. Thus, it is crucial to accompany the presentation of derived facts with *provenance information*, such as the set of derivation trees [2] of a given fact [4], [5]. These may be viewed as explanations for the derived facts.

However, presenting full provenance in such contexts is unlikely to be useful to the user. This is due to the complexity and large size of provenance information. To illustrate, the number of different rules that may be used to *directly* derive a single fact in AMIE exceeds 20; naturally, derivations based on these rules use additional facts, which again have many possible rules deriving them, etc. This means that the variability and quantity of derivation trees is extremely large in practical cases (there in fact may be *infinitely many trees* in presence of recursion in the Datalog rules). While compact representations for Datalog provenance were proposed in previous work [5], [4] (and used as intermediate, internal, representations in our context), their complex structure renders them unsuitable for presentation.

Our solution to this problem is based on the intuitive observation that not all derivation trees are equally "interesting". First, a derivation tree importance is effected by the *facts* that are used in the derivation ("supporting facts") as well as the fact that is eventually derived (we may be interested in explanations for some facts, or ones that are based on some facts). Second, it is effected by the *rules* that are used for derivation. Reconsidering our running example, we note that AMIE assigns *confidence scores* to the different rules. Intuitively, derivations that make extensive use of low-confidence rules, are less convincing and less useful for presentation purposes.

To this end, we present selP, a system that allows the selective, user-defined, tracking and presentation of provenance information. The input to the system, in addition to a Datalog program (set of derivation rules) and a relational Database, is a *specification of requested provenance*, in a novel query language called selPQL that we have designed for this purpose. selPQL is based on two facets, as follows. The first is the selection of a subset of derivation trees that are of interest. This subset may be specified by the user, via a notion of *derivation tree patterns*, which are reminiscent of tree patterns used in XML querying, modified so that nodes refer to (base or derived) facts. The second facet is the ranking of derivation trees, based on assigning weights to the different rules and aggregating them to form weights of derivation trees. Together, this defines the requested provenance as the *top-k derivation trees out of those conforming to the given pattern* (k, number of presented trees, is a user-defined parameter). We have designed a novel algorithmic solution to compute these top-k derivations.

selP may operate in one of two "modes". In an "online" mode, the selPQL query is specified *before* execution of the Datalog program, and is then interpreted as instructions to selP, specifying which parts of the provenance to track. This may be the case if we want to restrict provenance tracking in advance. An "offline" mode is suitable for a case where the selPQL query is specified *after* the program execution. Here

[1] Strictly speaking AMIE is based on Inductive Logic Programming, but derived rules may be translated to Datalog.

[2] Various other provenance models appear in the literature, see discussion of related work.

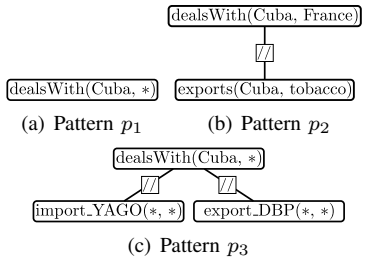
Country	Product
France	wine
Cuba	tobacco
Cuba	coffee beans

(a) exports

Country	Product
Cuba	wine
Mexico	wine
Mexico	tobacco
France	tobacco

(b) imports

Fig. 1: Database



(a) Pattern p_1 (b) Pattern p_2

(c) Pattern p_3

Fig. 2: Tree Pattern Examples

we track full provenance (stored using a compact, internal, representation), and use it as input to our novel top-k query evaluation algorithm. Offline evaluation leads to a storage overhead, but on the other hand allows issuing multiple queries without re-running the program.

We will demonstrate `selP` in the context of the AMIE system, using real-life data from YAGO [6] and real-life rules extracted by AMIE based on YAGO. YAGO is a large semantic knowledge base, derived from Wikipedia and additional sources, and its data and AMIE-derived rules range over multiple interesting domains. Specifically, we will demonstrate `selP` in the context of information regarding trading relationships between countries, academic relationships and influence, movies and geographic data. Using multiple examples, and interactively engaging the audience, we will demonstrate that `selP` allows users to obtain concise, useful, relevant and clear explanations for derived facts, even in settings where complete provenance information is incomprehensible.

Related Work: Data provenance has been studied for different data transformation languages, from the positive relational algebra to functional programs, and with different provenance models (see e.g. [7], [8], [4], [9]). Obviously there is a tradeoff between the expressivity of transformations for which provenance is captured, and the ease of viewing and understanding provenance. `selP` is unique in that it supports provenance for highly expressive data transformations (in Datalog), and handles the complexity of provenance through an expressive and intuitive query language for provenance, supporting in particular top-k queries. Previous solutions for querying provenance are either suitable only for weaker transformation models (see e.g. [10]) or are not declarative [7] and also lack a ranking mechanism. Last, works on querying *workflow provenance* for presentation (e.g. [11]), focus on the workflow's modules, inputs and outputs flow, and data provenance is typically abstracted away.

II. Model

We (informally) introduce the model underlying `selP`, through a running example.

A. Datalog

We refer the reader to [12] for a formal definition of Datalog and here we only illustrate it with our running example.

Example 2.1: The following three rules (focusing on international trade relations, and presented here using the Datalog syntax) were mined by AMIE [3] using the YAGO Knowledge Base. We note that since rules were mined by AMIE based on correlation in Data, the validity of some rules (and in particular r_1, r_2) is questionable.

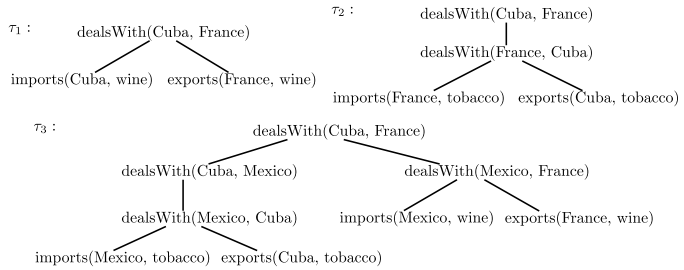
```
r1  dealsWith(a, b):- imports(a, c), exports(b, c)
r2  dealsWith(a, b):- dealsWith(a, f), dealsWith(f, b)
r3  dealsWith(a, b):- dealsWith(b, a)
```

The lefthand side of each rule (in the above example $dealsWith(a, b)$) is called the rule's *head*, and the righthand side is its *body*. Variables (e.g. a, b) are *bound* if they appear in the head, and are otherwise *free*. Rules are evaluated with respect to a instance I of the relations, binding variables to constants accordingly. Considering the instance I of Fig. 1, for the first rule we may assign a, b, c to $Cuba, France, wine$ resp. Values of bound variables for each such assignment are used in its head, possibly generating a new fact (e.g. $dealsWith(Cuba, France)$). The new fact is added to the instance, and evaluation continues until reaching a fixpoint.

B. Provenance

A *derivation tree* of a fact t with respect to a Datalog program and a Database instance I is a tree whose nodes are labeled by facts. The root is labeled by t, leaves are labeled by facts in I, and internal nodes by derived facts. The tree structure is dictated by the Datalog program: the labels set of the children of node N corresponds to an instantiation (via an assignment) of the body of some rule r, such that the label of N is the corresponding instantiation of r's head (we refer to this as an *occurrence* of r in the tree). Finally, the *provenance* of a fact t with respect to a program and an input instance is defined as the *set of possible derivation trees* of t [4]. Note that this set may be quite large, and in fact may be *infinite* when the Datalog rules are recursive.

Example 2.2: Three (out of the infinitely many) derivation trees for the fact $t = dealsWith(Cuba, France)$, based on the program given in Example 2.1 and the example Database given in Figure 1, are presented next:



C. Querying the Provenance

A *derivation tree pattern* is a node-labeled tree. Labels correspond to facts, that may include wildcards (*) instead of

values. Edges may be marked as *regular* (/) or *transitive* (//), and in the former (latter) case may be matched to a single edge (path of any length). We say that a derivation tree t satisfies a pattern p if there exists a homomorphism from nodes and (transitive) edges of p to nodes and edges (paths, respectively) of t, such that the root of p is mapped to the root of t, and node labels and descendant relations are preserved. We omit the formal definition and illustrate it with an example.

Example 2.3: Three derivation tree patterns are shown in Figure 2. Pattern p_1 will match any derivation tree of facts $dealsWith(Cuba, *)$ for some constant replacing the wildcard. p_2 queries the structure of derivations, specifying that we are only interested in derivations of $dealsWith(Cuba, France)$ that are (directly or indirectly) based on the fact that Cuba exports tobacco. Pattern p_3 is useful when (omitted) rules perform *integration* of two ontologies (YAGO and DBPedia). Intuitively p_3 will match derivations based on integrated data from both sources: imports (exports) information from YAGO (DBPedia).

Since there may still be (infinitely) many trees matching a pattern, we further *rank* derivation trees based on the rules and facts used in them. For that, we associate with each Database fact and each Datalog rule a numeric *weight*. Weights are elements of an ordered *commutative monoid*, which is an algebraic structure $(M, +, 0, <)$ where $+$ is an associative and commutative binary operation referred to as the aggregate function, 0 is an identity for $+$ and $<$ is a total order over the monoid elements.

Example 2.4: Each rule mined by AMIE is associated with a *confidence* value reflecting the rule "reliability". This serves as our weighting function, mapping rules to elements in the ordered monoid $([0, 1], \cdot, 1, <)$, e.g.: $w(r_1) = 0.8$, $w(r_2) = 0.5$, $w(r_3) = 1$. Individual confidence scores are in the range $[0, 1]$ (1 being the highest confidence), they are aggregated via multiplication, and are ordered based on standard order relation on numbers. Similarly, fact weights may reflect their confidence (in this simplified example we assign the neutral 1 to all facts).

We then define a derivation tree weight as the aggregation of weights of facts and rules occurrences in the tree, using the monoid $+$ operation as aggregation function.

Example 2.5: Consider the derivation trees given in Example 2.2 and the above weight-aware datalog example. The weights of the trees are $w(\tau_1) = 0.8$, $w(\tau_2) = 0.8 \cdot 1 = 0.8$ and $w(\tau_3) = 0.5 \cdot 1 \cdot 0.8 \cdot 0.8 = 0.32$

The combination of weighting and pattern (and k, number of requested results) is referred to as a `selPQL` query. Its *top-k results*, which are the system output, are the k derivation trees of highest weight, out of those satisfying the pattern.

Example 2.6: The top-2 query results for our running example (with pattern p_2) are τ_2 and τ_3 in Example 2.2 with weights of 0.8 and 0.32 respectively. Note that τ_1 does not match the pattern.

III. System Architecture and Evaluation

`selP` is implemented in JAVA with JAVAFX GUI using SceneBuilder, and runs on Windows 7. It uses MS SQL server as its underlying database management system. The system architecture is depicted in Figure 3, and the input/output user interfaces are depicted in Figures 4 and 5 respectively. As explained in the Introduction, `selP` can either operate in online or offline mode, based on the user choice. Evaluation of `selPQL` queries in both cases are based on three main components: (1) "intersecting" the pattern with the program/provenance, (2) compact representation of provenance, and (3) finding top-k trees based on this representation. We next briefly explain each component, omitting details for lack of space.

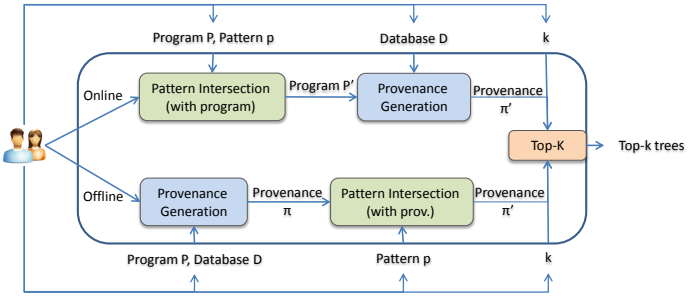


Fig. 3: System architecture

Pattern intersection (with program): This module implements a novel algorithm that given a Datalog program P (fed as input through the GUI presented in the lefthand side of Fig. 4) and a tree pattern p (see righthand side of Fig. 4), generates a new program P' which is the "intersection" of the program with the pattern. Namely, P' is such that (up to relation renaming) its derivation trees are exactly those derivation trees of P that satisfy p. This is done by introducing new relation names that reflect "guarantees" to generate gradually larger parts of the pattern, and rules to enforce that.

Provenance Generation: This module implements the algorithm of [4] to compute, *along side with standard evaluation of the Datalog program* a succinct, *internal* representation of (full) provenance for a given Datalog program and a given instance. The representation is through a system of fixpoint equations, with variables standing for facts and equations following the structure of the program (see [4]).

Top-K: Given the system of equations representing full provenance, this module applies a novel iterative algorithm that computes the top-k derivations. The basic idea is that at iteration i we compute, for every intermediate fact (represented by a variable in the equation system), its top-k derivations out of those of depth at most i; these are the "best" (highest-weight) extensions of those derivations computed up to iteration $i-1$. We may show that convergence to a fixpoint is guaranteed. The output trees are shown graphically, as depicted in Figure 5, alongside with the standard Datalog output.

Putting it all together: As depicted in Figure 3, the modules are combined for either online or offline evaluation. In the online mode, `selP` sequentially applies the three modules as follows: first, the pattern is intersected with the Datalog program, and the output ("intersected") program is fed to the provenance generator. The full provenance is then fed to the Top-K module, outputting the top-k derivation trees. Offline evaluation uses the modules in a different order: provenance

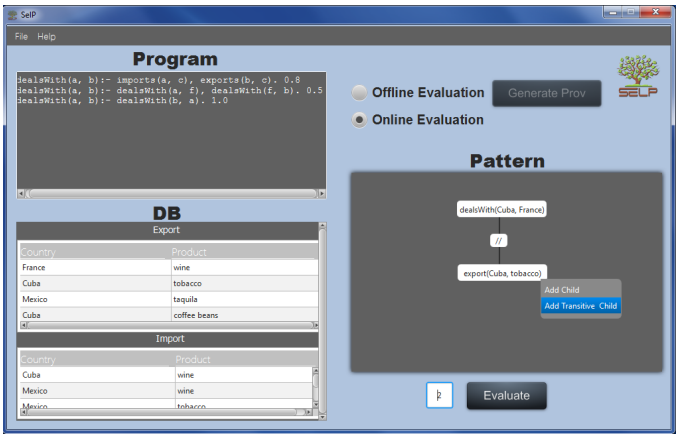


Fig. 4: Input screen

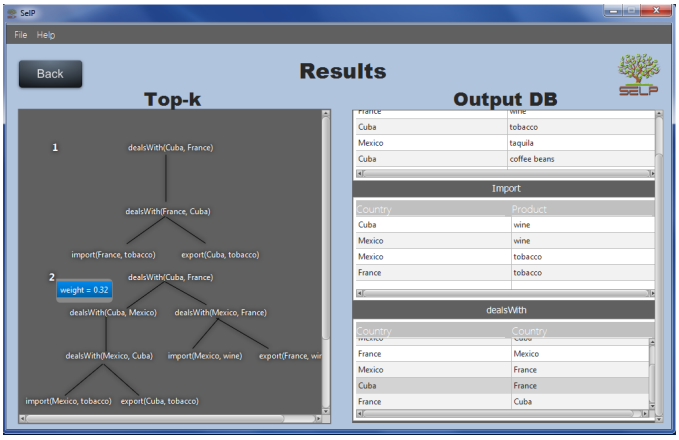


Fig. 5: Output screen

generation (along side with program execution) is applied with respect to the input program, generating full provenance. Then, the pattern is intersected w.r.t. the generated provenance rather than the program (through a modified intersection module). Finally, Top-K is applied as in the online mode.

IV. Demonstration Scenario

We will demonstrate that `selP` provides concise and useful explanations for the output of data-intensive applications. The system's operation will be demonstrated in an Information Extraction setting, using real-life data from YAGO (stored in a relational Database) and real-life weighted Datalog program, whose rules and their weights were extracted by AMIE. The demonstration will interactively engage the audience, demonstrating the different facets of the system.

For the first part of the demonstration we will run in advance the Datalog program with full provenance tracking (as in the offline approach). We will ask participants to choose a "topic", out of a variety of options available in the YAGO dataset: trading data (as in our running example), academic and influence relationships, movies and geographic data. For the selected domain, we will browse through the relevant parts of the database and rules, focusing on some examples and showing them to the audience. We will then ask the audience to choose a few derived facts, and to "guess" some intuitive explanations for their derivations. Next, we will pose the appropriate simple patterns (each selecting explanations for one of the chosen facts), and use `selP` to compute the top-3 explanations for the facts. We will present the explanations to the audience, demonstrating that they intuitively reflect "most prominent" explanations for the reason the facts of interest were obtained, and discussing whether the presented explanations match their original intuition.

Second, we will ask the audience to further specify `selPQL` queries with respect to the same demonstrated facts. For instance, if they were expecting a particular supporting fact to show up in prominent explanations, they may wish to explicitly ask for explanations using this fact. The queries will be fed by the participants to `selP` through its user-friendly GUI, and we will again show and discuss the analysis results.

Third, we will demonstrate the online approach, using the same `selPQL` queries and applying the system's online evaluation. We will demonstrate that the overhead of selective provenance tracking is feasible and significantly reduced with respect to full provenance tracking.

Finally, we will allow the audience to look "under the hood". In particular we will show the "instrumented" Datalog program generated as part of the online evaluation algorithm. We will also show the intermediate internal representation for the full provenance, explaining why full provenance cannot be presented as intuitive and concise explanations. We will highlight how our approach of presenting selective provenance allows to alleviate this difficulty.

Acknowledgments: This research was partially supported by the Israeli Ministry of Science, by the Isareli Science Foundation (ISF), by the Broadcom Foundation and Tel Aviv University Authentication Initiative, and by the Advanced ERC grant Modas (grant 291071).

References

[1] B. T. Loo and W. Zhou, *Declarative Networking*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[2] R. Ronen and O. Shmueli, "Soql: A language for querying and creating data in social networks," in *ICDE*, 2009, pp. 1595–1602.

[3] L. A. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek, "Amie: association rule mining under incomplete evidence in ontological knowledge bases," in *WWW*, 2013, pp. 413–422.

[4] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*, 2007.

[5] D. Deutch, T. Milo, S. Roy, and V. Tannen, "Circuits for datalog provenance," in *ICDT*, 2014.

[6] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A Core of Semantic Knowledge," in *WWW*, 2007.

[7] U. A. Acar, A. Ahmed, J. Cheney, and R. Perera, "A core calculus for provenance," *Journal of Computer Security*, vol. 21, no. 6, 2013.

[8] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom, "Trio: A system for data, uncertainty, and lineage," in *VLDB*, 2006.

[9] D. Gawlick and V. Radhakrishnan, "Fine grain provenance using temporal databases," in *TaPP*, 2011.

[10] G. Karvounarakis, Z. G. Ives, and V. Tannen, "Querying data provenance," in *SIGMOD Conference*, 2010, pp. 951–962.

[11] O. Biton, S. C. Boulakia, and S. B. Davidson, "Zoom*userviews: Querying relevant provenance in workflow systems," in *VLDB*, 2007.

[12] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.