PODIUM: Probabilistic Datalog Analysis via Contribution Maximization

Tova Milo, Yuval Moskovitch, and Brit Youngmann
Tel Aviv University
{milo,moskovitch1,brity}@post.tau.ac.il

ABSTRACT

The use of probabilistic datalog programs has been advocated for applications that involve recursive computation and uncertainty. While using such programs allows for a flexible knowledge derivation, it makes the analysis of query results a challenging task. Particularly, given a set O of output tuples and a number k, one would like to understand which k-size subset of the input tuples has *affected the most* the derivation of O. This is useful for multiple tasks, such as identifying critical sources of errors and understanding surprising results. To this end, we formalize the Contribution Maximization problem and present an efficient algorithm to solve it. Our algorithm injects a refined variant of the classic Magic Sets technique, integrated with a sampling method, into top-performing algorithms for the well-studied Influence Maximization problem. We propose to demonstrate our solution in a system called PODIUM. We will demonstrate the usefulness of PODIUM using real-life data and programs, and illustrate the effectiveness of our algorithm.

KEYWORDS

Probabilistic Datalog; Results Explanations

ACM Reference Format:
Tova Milo, Yuval Moskovitch, and Brit Youngmann. 2019. PODIUM: Probabilistic Datalog Analysis via Contribution Maximization. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM'19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3357384.3357841

1 INTRODUCTION

Real-life applications often rely on an underlying database in their operation. While many of these applications employ convectional SQL as their query language, the use of *probabilistic datalog* has been recently advocated for applications that involve recursive computation and uncertainty, including e.g., network management [11], and information extraction [5]. To illustrate, consider AMIE [5], a system that mines logical rules from Knowledge Bases (e.g., YAGO [12]), based on correlations in the data. The mined rules, treated as a datalog program, are evaluated w.r.t. a KB, e.g., to address data incompleteness. Since the rules are automatically mined, there is an inherent uncertainty w.r.t. their validity. AMIE thus associates probabilities to the rules, yielding a probabilistic datalog program.

exports

Country	Product
France	wine
France	vinegar
France	oil
Cuba	tobacco
Cuba	sugar
Cuba	nickel
Russia	gas

imports

Country	Product
Germany	wine
USA	vinegar
Pakistan	oil
India	tobacco
Denmark	sugar
Iran	nickel
Ukraine	gas

dealsWith (edb copy)

Country	Country
France	Cuba

Table 1: Example Database.

The use of probabilistic datalog allows for a flexible and expressive knowledge derivation, yet introduces intricate relations between the data items. This makes the analysis of query results a challenging task. In particular, given a set O of output tuples of interest, one would like to understand which k-size subset of the input tuples have *affected the most* the derivation of O. This is useful for multiple tasks, such as identifying critical sources of errors and understanding the reasons for surprising results [6, 9]. Acquiring information about the most influential tuples may also serve users in obtaining explanations for output tuples using selective provenance tracking systems such as [4]. These systems provide explanations based on user-defined patterns, however, defining the patterns may be challenging without prior knowledge on the tuples that have contributed the most to the results.

To illustrate our problem consider the following example.

Example 1.1. *Consider a probabilistic datalog program consisting of* 3 *rules, mind by AMIE over YAGO. A sample database instance is depicted in Table 1. This program outputs a binary relation dealsWith, including information on international trade relationships.*
r_1(0.8) `dealsWith(a, b):- dealsWith(b, a)`
r_2(0.7) `dealsWith(a, b):- exports(a, c),imports(b, c)`
r_3(0.5) `dealsWith(a, b):- dealsWith(a, f), dealsWith(f, b)`
Focusing on a set of derived facts of interest (e.g., `dealsWith(USA, Iran)`, `dealsWith(Pakistan, India)` `dealsWith(Russia, Ukraine)`*), one may wish to understand which database facts have led to this inference. Presenting all tuples that took part in the computation as an explanation may be cumbersome and not informative. For instance, nearly* 36% *of the database tuples are used to derive solely the fact* `dealsWith(USA, Iran)`. *Thus, it is of great importance to identify a* small set *of facts that have* contributed the most *to this inference.*

To facilitate such an analysis, one needs first to formally *quantify the notion of contribution*, while considering the following three issues: First, as opposed to the simple SQL setting, the *recursive relationship* between input and output data, and the *uncertainty* induced by the rules' probabilities need to be considered. For instance, in the above recursive program, the rules' probabilities (in parenthesis) model the fact that transitive trade relations are considered less trustworthy than direct relations. Second, note that selecting the top k tuples with the highest individual contribution is not the

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11.
https://doi.org/10.1145/3357384.3357841

same as finding a k-size set with highest overall contribution, as two input tuples may contribute to exactly the same derived facts. Therefore, one must consider the *joint* contribution of a set of input tuples to a set of output tuples. Our definition, which considers such joint contribution of sets of tuples, ensures a greater coverage of the target tuples of interest. Last, the rules' probabilities are independent, and so are the instantiations of each rule. Thus, the *independence* of a single rule instantiation from other derivations in the program needs to be considered as well. For instance, among many other tuples, the tuple t_1 = `dealsWith(France, Cuba)` takes part in the derivation of t_2 = `dealsWith(USA, Iran)`. To properly asses its own contribution to t_2, we focus in our definition on the marginal contribution of t_1 (or more generally, of a k-size set of tuples of interest), regardless of other parts in the derivation.

To this end, we formalize the Contribution Maximization (CM) problem of finding, among a set of database tuples of interest T, a bounded-size subset with the maximal contribution to a target set of output tuples O. Interestingly, a similar problem was studied in the context of social networks analysis. Particularly, the classic Influence Maximization (IM) problem [7] is the problem of finding a bounded-size set of influential users in a social network (called a seed-set), so that their aggregated influence (which may propagate transitively in the network) on other users is maximized. Reducing our problem to this well-studied problem allows the use of highly efficient algorithms developed for IM. However, as we explain, a naïve such approach yields prohibitively expensive algorithm, in terms of both memory consumption and running times. To overcome this, we propose an optimized algorithm which injects a refined variant of the classical Magic Sets technique [2], integrated with a sampling method, into the (adapted) IM algorithms. We have implemented our solution in a system called PODIUM (PrObabilistic Datalog contrIbUtion Maximization), which we will demonstrate here.

Related Work. There is a wealth of work on query results explanation. One line of works utilizes *data provenance* [4, 8, 10], however, provenance for recursive datalog is known to be large and often impractical to materialize. Another line of works points, like us, on database tuples that have significantly affected the results [6, 9, 10]). However, those works have mainly focused on *non-recursive* SQL queries, very often *disregarding probabilistic inference*. Furthermore, they have only focused on quantifying the contribution of a *single* database tuple to a *single* output tuple. Among the variety of works, our approach is most similar to the work presented in [9], where the notion of *responsibility* for non-recursive SQL queries was defined (for a single input/output tuple). While their relational SQL setting can be seen as a restricted version of ours, a key difference is that we aim to quantify the *marginal contribution* of input tuple(s) to output tuple(s) *independently of other derivations in the program*, whereas they focus on dependencies. Our work and theirs are thus complementary, providing two alternative angles of involvement in the result computations.

2 TECHNICAL BACKGROUND

We start by providing a brief overview of (probabilistic) datalog and the IM problem, then present the CM problem and our optimized algorithm. For space constraint, proofs and formal definitions are deferred to our technical report [14].

2.1 Preliminaries

Probabilistic Datalog. We refer the reader to [2] for a formal definition of (probabilistic) datalog and here we only illustrate it with the example presented in the Introduction.

Consider again the probabilistic datalog program depicted in Example 1.1, which outputs the binary relation dealsWith (an edb "copy" of this relation appears as well, with rules for copying its content that are omitted for brevity). The probabilities, in parenthesis, model the fact that transitive trade relations are considered less trustworthy (likely to happen) than direct relations[1].

We refer to a probabilistic datalog program as a pair (P, w) where P is a set of rules and w is a function assigning probabilities to them. The interpretation of a rule r's probability is that we trust a given instantiation of r with probability of $w(r)$. The lefthand side of a rule is called the rule's head, and the righthand side is its body. Rules are evaluated w.r.t. a database. Consider the database in Table 1, for the rule r_2 we may assign the variables a, b, c the values `France, Germany ,wine`, resp. Each rule instantiation $r(inst)$ may generate a new fact (e.g., `dealsWith(France,Germany)`). New facts are added to the program output, denoted as $P(D)$, and the evaluation continues until reaching a fixpoint. In the derivation process of a probabilistic program, for each rule instantiation *that has not been considered yet*, we draw if it should be fired according to the rule's probability. Only instantiations for which the result of the draw was positive are fired.

We distinguish between extensional (edb) and intentional (idb) database facts. The former are the input facts while the latter are derived by the program. It is common to describe the process of datalog evaluation through the notion of *derivation trees* [4, 8, 10]. A derivation tree of a tuple t is a finite tree whose root is labeled by t, the leaves by edb facts, and the internal nodes by idb facts. The children of a node n correspond to an instantiation of a rule r. For example, consider the subgraph within the dashed part of Figure 1, representing a derivation tree of `dealsWith(USA,Iran)`.

Influence Maximization (IM). As mentioned, we employ concepts from the classic IM problem for our Contribution Maximization problem. Let $\mathcal{G} = (V, E, W)$ be directed weighted graph models a social network, where V is the set of nodes (users) and each edge $(u, v) \in E$ is associated with a weight $W(u, v) \in [0, 1]$. Given a function $I(\cdot)$ dictating how influence is propagated in the network and a number k, IM is the problem of finding a seed-set of users $S = argmax_{T \subseteq V, |T|=k} I(T)$, where $I(T)$ denotes the expected number of nodes influenced by T. The function $I(\cdot)$ is defined by a propagation model. We will employ here the common *Information Cascade* (IC) model, which is compatible with most IM algorithms [3, 13].

The influence of a seed-set under the IC model is estimated as follows. Independently, for each edge (u, v) with the weight $W(u, v)$ we flip a coin indicating whether this edge is active or not. The edges in $\mathcal{G}$ for which the coin flip indicated an activation will be successful are declared to be live; the remaining edges are blocked. Fixing the outcomes of the coin flips and initially activating a seed set S, a node v ends up influenced if there is a path from some node in S to v consisting entirely of live edges.

[1] Since probabilities on database tuples can be modeled by probabilistic rules, we consider here only rules probabilities.

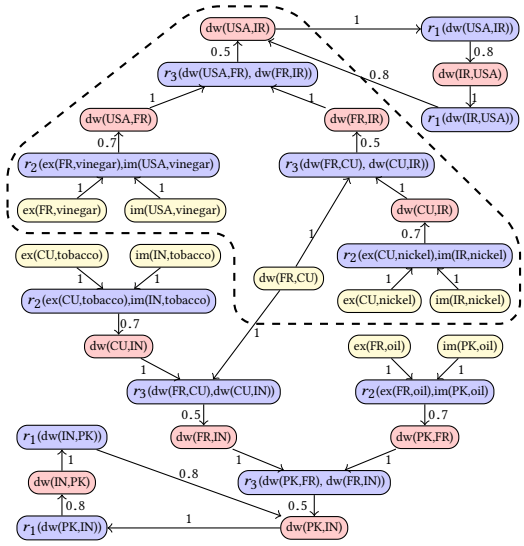

Figure 1: A partial WD graph.

Recent IM algorithms are based on the Reverse Influence Sampling (RIS) approach [3], which samples nodes independently and uniformly, then for each sampled node, constructs a Reverse Reachability (*RR*) set consisting of its sampled sources of influence. Next, k nodes that maximize the number of covered *RR* sets are selected.

2.2 Problem Formulation

We next formalize the notion of *tuples contribution* and define the Contribution Maximization problem.

Given a probabilistic datalog program, consider all its possible executions, and recall that in each such execution every instantiation of a rule $r(inst)$ is drawn to fire with a probability of $w(r)$. We devise the Weighted Derivation (WD) graph, a directed graph that captures all these possible executions, integrating all derivation trees of the program by merging their common parts.

More formally, let (P, w) be a probabilistic datalog program and D a database instance. The corresponding WD graph $\mathcal{G} = (V, E, W)$ is defined as follows. There is a distinct node per each edb tuple, each idb tuple, and each rule instantiation. For every rule instantiation $r(inst) = h\text{:-}b_1, \ldots, b_n$, the node $r(inst)$ has n incoming edges $(b_i, r(inst))$, $i = 1 \ldots n$, from the edb/idb facts in its body, and an outgoing edge $(r(inst), h)$ to the idb fact h in its head. Each edge of the form $(r(inst), h)$ is assigned with the rule's weight $w(r)$, and all other edges have a weight of 1.

Example 2.1. *Continuing with our running example, Figure 1 depicts a partial WD graph. Here the idbs are colored in red, the edbs in yellow, and the auxiliary rule instantiations nodes are colored in purple. Observe that two derivation trees of the idbs* `dealsWith(USA,Iran)`, `dealsWith(Pakistan, India)` *have been merged.*

Intuitively, a tuple $t_1 \in D$ contributes to a tuple $t_2 \in P(D)$ if the WD graph contains a path from t_1 to t_2. To quantify the marginal contribution of the derivations involving a set of tuples $T \subseteq D$ to a set of tuples $O \subseteq P(D)$, *independently of other derivations in the graph*, we consider a random draw of fire-or-not for all rules instantiations (edges) in the graph. We measure the contribution as the expected number of tuples from O reachable from tuples in T in such a randomly generated subgraph. Formally, given the WD graph $\mathcal{G}$ of a database D and a program (P, w), let g be a random subgraph generated from $\mathcal{G}$ as described above. We define the contribution of a set $T \subseteq D$ to a set $O \subseteq P(D)$, denoted $c(T \rightsquigarrow O)$, as the expected number of nodes in O reachable from nodes in T in the subgraph g.

Observe that this definition captures the natural properties expected from a contribution function, as discussed in the Introduction. Namely, the recursive relations and the uncertainty are captured via the WD graph. The more paths from T to O that the WD graph contains and the higher the probabilities of rules they include, the greater is the contribution of the set T to the set O. Moreover, our measure captures the involvement of tuples in T in the derivation of the tuples in O, independently of other parts of the derivations. This implies that we capture the contribution of T to O, regardless of the probability of the other facts.

We are now ready to formally define the CM problem.

Definition 2.2 (CM). *Given a probabilistic datalog program (P, w), a database D, a set $T \subseteq D$, a set $O \subseteq P(D)$, and a number $k \leq |D|$, find a k-size set $S \subseteq T$ s.t: $S = argmax_{S' \subseteq T} E[c(S' \rightsquigarrow O)]$*

Example 2.3. *Continuing with our example, let $T = D$, $O =$* {`dealsWith(USA,Iran)`,`delasWith(Pakistan, India)`, `delasWith(Russia, Ukraine)`}*, and $k = 2$. Observe that the tuple* `dealsWith(France,Cuba)` *is involved in the derivations of both the tuples* `dealsWith(USA,Iran)`, `dealsWith(India,Pakistan)`, *while all other database tuples are a part of the derivation of a single derived fact. Intuitively, to maximally contribute to all facts in O, we need to pick one edb fact that is a part of the derivation of* `delasWith(Russia, Ukraine)` *(i.e.,* `exports(Russia, gas)`, `imports(Ukraine, gas)`*) and the fact* `dealsWith(France,Cuba)`. *Indeed, any such pair yields the maximal contribution score.*

2.3 Algorithms

Naïve algorithm. A key observation is that our contribution function matches nicely the IM influence function. Indeed, CM can be formulated as a variant of IM. Therefore, as we show in [14], existing IM algorithms can be adjusted to solve CM. A naïve algorithm would therefore be to first build the WD graph, then to run an (adjusted) IM algorithm over it. However, our experimental results show that the WD graph may be prohibitively large, and thus this naïve algorithm, referred to as NaïveCM, is impractical.

Optimized algorithm. We therefore devise an optimized algorithm, referred to as *Magic*^{S}CM, that achieves a significant saving in both memory and running time by employing two optimizations.

Our first optimization harnesses principles from the classic Magic Sets transformation [2], to construct only the graph parts that are essential for the computation of the (adjusted) IM algorithm. We refer to our algorithm with only this Magic Sets optimization as the *Magic*CM algorithm. In short, recall that NaïveCM constructs the full WD graph, then employs an IM algorithm, which constructs for each sampled idb node its corresponding *RR* set. In the *Magic*CM algorithm, rather than constructing the full WD graph $\mathcal{G}$, we build *on-the-fly*, for each sampled node, only the subgraph of $\mathcal{G}$ that is relevant to its *RR* set computation. For that we consider, for each sampled tuple t, a top-down evaluation of the program P with the query $q = t$. We apply the Magic Sets transformation to obtain a

new program P_t^m that computes only the relevant facts, then assign probabilities to the rules, obtaining a new program (P_t^m, w_t^m). The key challenge here is assigning probabilities to the rules of the transformed program s.t. the new WD graph is isomorphic to the desired subgraph of the original WD graph.

Our second optimization refines the above process by bundling the subgraph construction with the sampling done by the IM algorithm. We refer to *Magic*CM enhanced with this optimization as *Magic*^{S}CM. In short, we utilize the fact that in the IM algorithm the *RR* set computation contains a further sampling process that considers only a *sampled subgraph* of $\mathcal{G}$. We can thus incorporate a sampling step in the subgraph construction, building a significantly smaller subgraph of the WD graph that includes only such sampled edges. The key challenge here is to carefully inject the IM sampling into the Magic-Set graph construction, yielding a subgraph that is still isomorphic to the relevant part of the original WD graph.

3 SYSTEM AND DEMONSTRATION

Implementation. PODIUM is implemented in JAVA 8 by extending IRIS [1], a JAVA-based system for datalog evaluation. The (adjusted) IM algorithm our prototype employs is IMM [13]. The user interacts with the system using a dedicated user interface, implemented in HTML5/CSS3, depicted in Figure 2 (see details below).

Demonstration. We demonstrate the usefulness of PODIUM using real-life data extracted from YAGO, and probabilistic datalog programs, whose rules and their probabilities were mined by AMIE. The participants will be asked to play the role of data analysts, examining the benefits of PODIUM in analyzing the output of data-intensive applications. We also provide a "behind the scenes" view of the system, demonstrating the effectiveness of our optimizations.

We begin the demonstration by presenting the capabilities of PODIUM using multiple examples in different domains, such as trading data, academic influence relations, movies and geographic data. Using PODIUM UI, the participants will first pick a domain (i.e., tables and a datalog program). See, for example, the upper part of Figure 2, where the chosen domain is trading data. We will then browse through the relevant tables (here, the `Exports, Imports` and `DealsWith` tables) focusing on intriguing derived facts, and use the UI to select sets of input/output tuples of interest. The selection is done by pointing on the selected tuples, or alternately, inserting simple patterns specifying the tuples of interest. For example, in Figure 2, using patterns, we have chose the set of input tuples to be all tuples stating that some country is exporting uranium (using the pattern `Exports(*,Uranium)`), and have manually pointed on several output tuples (e.g., `DealsWith(United States,North Korea)`). Advanced users can specify the tuples of interest using general SQL queries on the input/output tables via a text-based editor, using the *advanced* screen. Next, we show that the number of tuples involved in the computation is indeed very large (which motivates focusing on a small set of influential tuples), then let the user choose how many tuples they want to focus on (by setting k). The set of k most influential edb tuples and their expected contribution scores, are then displayed on a results page (omitted from presentation for space constraint). To visualize the tuples' contribution, we further display the relevant parts of the WD graph and highlight the derivation paths on it.

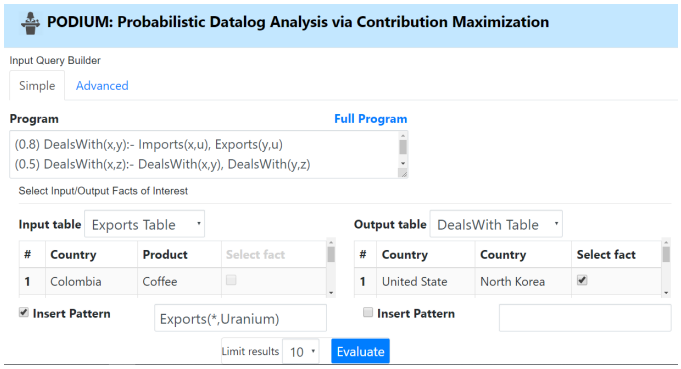

Figure 2: PODIUM UI: Input Builder.

Next, we will illustrate to the audience the running time and memory reduction of our optimized algorithm, compared with the naïve approach. For this part of the demonstration, we will use growing fragments of the underlying database, showing the limitations of NaïveCM to scale. We will run PODIUM in three modes, employing our three algorithms: NaïveCM, *Magic*CM, and *Magic*^{S}CM. We will present the running times and memory consumption of each algorithm, illustrating the effect of each optimization. For example, we will show that, on average, *Magic*CM and *Magic*^{S}CM achieve each a memory reduction of 2 and 3 orders of magnitude, resp. compared with NaïveCM (see further experimental results in [14]).

Acknowledgments. This work has been partially funded by the Israeli Ministry of Science, Technology and Space, the Israel Innovation Authority, the Israel Science Foundation, the Binational US-Israel Science foundation, Len Blavatnik, the Blavatnik Family foundation and the Deutsch family foundation.

REFERENCES

[1] IRIS Reasoner. http://www.iris-reasoner.org. (????).
[2] S. Abiteboul, R. Hull, and V. Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[3] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *SODA*.
[4] Daniel Deutch, Amir Gilad, and Yuval Moskovitch. 2015. Selective Provenance for Datalog Programs Using Top-K Queries. *PVLDB* (2015).
[5] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*.
[6] Bhargav Kanagal, Jian Li, and Amol Deshpande. 2011. Sensitivity Analysis and Explanations for Robust Query Evaluation in Probabilistic Databases. In *SIGMOD*.
[7] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the Spread of Influence Through a Social Network. In *SIGKDD*.
[8] Seokki Lee, Bertram Ludäscher, and Boris Glavic. 2019. PUG: a framework and practical implementation for why and why-not provenance. *VLDB J.* 28, 1 (2019).
[9] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F Moore, and Dan Suciu. 2010. The complexity of causality and responsibility for query answers and non-answers. *PVLDB Endowment* (2010).
[10] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *SIGMOD*.
[11] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. 2016. Big data analytics with datalog queries on spark. In *SIGMOD*. ACM.
[12] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*. ACM.
[13] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *SIGMOD*.
[14] Milo Tova, Moskovitch Yuval, and Youngmann Brit. 2019. Technical Report. (2019).