Short Paper

On Answering Why-Not Queries Against Scientific Workflow Provenance

Khalid Belhajjame
Université Paris-Dauphine, PSL Research University, CNRS, [UMR 7243], LAMSADE
Paris, France
Khalid.Belhajjame@dauphine.fr

ABSTRACT

Why-not queries help scientists understand why a given data item was not returned by the executions of a given workflow. While answering such queries has been investigated for relational databases, there is only one proposal in this area for workflow provenance, viz. the Why-Not algorithm. This algorithm makes the assumption that the modules implementing the steps of the workflow preserve the attributes of the input datasets. This is, however, not the case for all workflow modules. We drop this assumption, and show in this paper how the Web can be harvested to answer why-not queries against workflow provenance.

1 INTRODUCTION

Scientific workflows have been shown to facilitate and accelerate scientific data exploration and analysis in many areas of sciences [6]. A workflow can be viewed as an acyclic graph in which the nodes are modules that can be executed locally or remotely, and the edges specify the data dependencies between the constituent modules. Workflows have been utilized to model and enact in-silico experiments in a range of scientific fields, including proteomics, transcriptomics, metabolics, plant phenotyping, astronomy, and bio-medicine.

Fig. 1 illustrates an example of a simple workflow used for identifying the pathway associated with a given input metabolite (compound). Given a compound identifier, the first module returns a compound name, which is used to feed the second module to obtain the corresponding pathway.

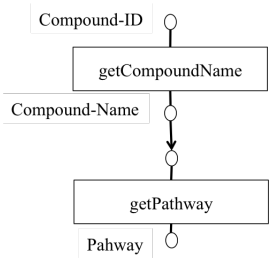

Figure 1: Example workflow.

Major workflow systems are instrumented to capture provenance information that records the data items used and generated by the workflow modules together with information specifying the lineage of such data items across the workflow execution.

Workflow provenance information can be utilized in a range of applications [8]. For example, it can be used to i)- estimate the quality of data based on the source data, ii)- determine the entity (author) to whom a given data item should be attributed, and iii)- debug errors in the workflow execution. In this work, we focus on an application that has received little attention within the workflow provenance community, namely why-not queries. Given the provenance traces of workflow executions, the scientist may want to understand why a given result, e.g., his/her favorite protein, does not appear in the result of workflow executions.

1.1 Related Work

We distinguish between two classes of proposals for answering why-not queries: instance-based and module-based.

Instance-based. Proposals that fall into this category attempt to find the data items in the inputs that are responsible for the non appearance of a given data item in the result. More specifically, they determine the changes that need to be applied to the input datasets for the data item provided by the user to appear in the result. Artemis [9] and Missing-Answers [10] are examples of algorithms for processing instance-based why-not queries over relational databases. There is no proposal in the state of the art that investigates instance-based why-not queries for workflow provenance.

Module-based. Proposals in this class attempt to identify the modules that are responsible for the non-appearance of a given data item in the workflow results. The only proposal in this category for workflow provenance is the Why-Not algorithm proposed by Chapman and Jagadish [4]. Indeed, while why-not queries have been investigated in databases, where datasets are manipulated using (white boxes) query operators with known operational semantics, this is not the case for scientific workflows, where the steps of the workflow are implemented by black boxes, the behavior of which is not necessarily known. Using the Why-Not algorithm proposed by Chapman and Jagadish, the user query is expressed as a set of atomic predicates that are combined using AND and OR. An atomic predicate is evaluated over a single attribute of the Input datasets of the workflow. Chapman and Jagadish make the assumption that the attributes of the input datasets are preserved by the modules that compose the workflow. This is not the case, however, in the general case. For example, the modules in the workflow illustrated in Fig. 1 do not preserve the attribute of the input, viz. $\texttt{Compound} - \texttt{ID}$, in that the output of the first and the second module do not contain information about the compound identifier.

As well as identifying the reason why a result is missing, a number of proposals investigated changes that can be made to the query to include known missing results (see e.g., [1, 5]).

In this work, we focus on explaining why a data item is missing from the results of a data-driven workflow. In doing so, we drop the assumption made by Chapman and Jagadish, and propose a solution that can be utilized for answering why-not queries for workflow with modules that do not preserve attributes of the input datasets. Furthermore, unlike the Why-Not algorithm which is module-based, our proposal is hybrid in that it seeks

to answer instance- and module-based why-not queries. This makes our solution the only hybrid solution targeted for workflow provenance.[1]

In the rest, we start by lying down the foundations in Sect. 2. We sketch our algorithm in Sect. 3. We then focus on operations that are central to our algorithm, namely determining if a module is picky, i.e., responsible for the non-appearance of an output data item, and determining the missing input data item for a given module in Sect. 4. Finally, we report on the results of a feasibility study that we conducted, and conclude the paper in Sect. 5.

2 FOUNDATIONS

We define a workflow WF as a directed acyclic graph in which the nodes correspond to modules, and the edges specify data flow dependencies. A data link connecting a module M_1 to a module M_2 specifies that the output produced by the invocation of the first is used as input to feed the execution of the latter.

The invocation of a module M, which we call module instance and denote by m, takes one or more data items as input and produce a data item as output. We write $m(d^{I_1}, \ldots, d^{I_n}) = d^O$, where $I_1, \ldots, I_n$ represents the domain of values of the inputs of M and O the domain of values of the output. For the purpose of this work, we consider that the modules of a workflow are functions, in that they return the same output data item given the same input data items.

The execution of a workflow WF gives rise to a workflow instance wf, which takes one or more data items and produce as a result a data item. Major scientific workflow systems are instrumented to capture provenance information of workflow instances specifying the data items used and generated by the workflow, which can be utilized to track the lineage of the workflow results.

Typically, a scientist would execute a workflow WF multiple times, and then proceed to the exploration and analysis of the results. We are, therefore, interested in querying the datasets used and generated as a result of multiple executions (instances) of WF. We write $WF(D^{I_1}, \ldots, D^{I_n}) = D_O$ to denote that collectively the instances of the workflow WF, took as input the datasets $D^{I_1}, \ldots, D^{I_n}$ and generated the dataset D_O. Similarly, we write $M^{WF}(D^{I_1}, \ldots, D^{I_n}) = D_O$ to denote that the invocations (instances) of the module M that took place with the instances of the workflow WF used the datasets $D^{I_1}, \ldots, D^{I_n}$ and generated the dataset D_O.

Inverse of a Module. Central to our solution is the notion of the inverse of a module. The inverse of a module M, which we denote by M_{inv} takes as input data items that are type-compatible with the output of M, and delivers data items that are type-compatible with the inputs of M .

Picky module. A module M is picky with respect to a data item d if M_{inv} does not accept d as input. More specifically, M_{inv} throws an illegal input exception when its execution is fed d.

3 ANSWERING WHY-NOT QUERIES

3.1 Why-Not Queries

A user specifies a why-not query by specifying a data item, which has the same data type as the output of the last module of the workflow. Let such a module be $M^{WF}(D^{I_1}, \ldots, D^{I_n}) = D_O$ Consider for the sake of simplicity that such a module output data items that are characterized by the attributes $\langle a_1, \ldots, a_m \rangle$. A why not-query is a data item $\langle v_1, \ldots, v_m \rangle$ that was not generated by M when invoked within the workflow WF. That is, $\langle a_1, \ldots, a_m \rangle \notin D_O$. We denote such a data item in what follows by $d_{why-not}$. Note that, in the general case, a workflow may have multiple final modules. Our solution is applicable to those workflows. To do so, our algorithm is applied to each output data item (why-not query) provided by the user.

[1] There are existing hybrid solutions, but they are targeted for relational databases (see e.g., [2]).

3.2 Processing Why-Not Queries

To answer a why-not query, the modules of the workflow are explored from the sink to the source in a breadth-first fashion. To do so, we group the workflow modules into levels as illustrated in Figure 2. A module belongs to a level i if its outputs are connected to modules in levels $\leq i - 1$. Of course, this does not apply to the modules in level 0, the outputs of which carry the results of the workflow execution. The modules of a given level can be examined only if the examination of the modules of the previous level has completed.

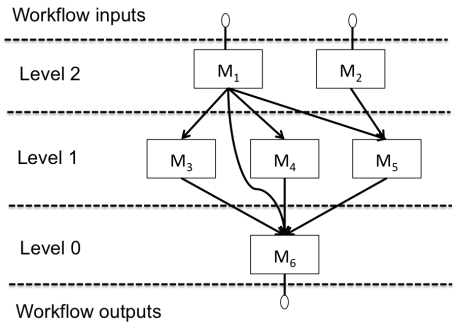

Figure 2: Workflow levels.

Algorithm 1, which we named Why-Not Detective, sketches the evaluation of a why-not query. It takes as input a data item $d_{why-not}$ specified by the user, and the workflow modules organized into levels starting from the sink WF_Modules. The modules of each level are examined to identify if the module is picky. Specifically, the inverse of the module in question M is examined to check if:

- It does not accept the corresponding data items that were generated by the inverse of the modules in the previous level. By corresponding data items, we mean data items generated by module parameters in the previous level, if any, that are connected to the output of the M module. The module M is flagged in this case as picky.
- It accepts the corresponding data items that were generated by the inverse of the modules in the previous modules. In this case, the data items the inverse of M produces are saved to be used to feed the inverse of the modules in the succeeding levels, if any. Notice here that the provenance of the workflow subjects to analysis will not contain such data items, otherwise, the workflow would have produced $d_{why-not}$.

The algorithm iterates over the modules of each level until i)- it finds modules of a given level that are picky, or ii)- it does not find any picky modules. In the first case, the algorithm stops (see line 15). Indeed, the modules of the succeeding level, or at least subset thereof, cannot be examined. This is because we will not have data items to probe the inverse of such modules with. The picky modules are therefore returned by the algorithm as the source of the non appearance of $d_{why-not}$ in the results. Note here that

our notion of picky module corresponds to the notion of *Frontier Picky Manipulation* in the Why-Not algorithm by Chapman and Jagadish [4]. In the second case (ii), the datasets used as input to the execution of the workflow are designated as the source of the non-appearance of $d_{why-not}$ in the results. Specifically, the data items returned by the inverses of the modules in the last level are designated as missing in the inputs datasets.

Example 3.1. Consider our example workflow in Fig. 1, and suppose that the scientists did not find the pathway identified by `map00220` in the results of the provenance of the workflow. In this simple example, we have two levels composed of one module each. The algorithm starts by examining the `getPathway` module. The inverse of this module accepts the value `map00220` and provides the compound name `L − Argentine`. This compound name is accepted by the inverse of `getCompoundName`, which in turn delivers the compound identifier `cpd : C00062`. The Why-Not detective concludes, in this case, that none of the modules is picky, and that the non appearance of the pathway `map00220` in the results is due to the missing input data item `cpd : C00062`.

Algorithm 1 Why-Not Detective

Input: $WF\_Modules = \{L_0, \ldots, L_k\}$ *//workflow modules grouped into levels (breadth) from the sink to the source.*
$d_{why-not}$ *// missing workflow result.*
Output: `PickyModules` *// set of picky modules*
`MissingInputData`*// set of data items missing in the workflow input dataset*

```
1:  PickyModules = ∅
2:  MissingInputData = ∅
3:  D_current = {d_why−not}
4:  D_current+1 = ∅
5:  for Level in WF_Modules do
6:      for M in Level do
7:          if accept(inv(M), getInput(D_current)) then
8:              data_current+1 += getResult(inv(M), getInput(D_current))
9:          else
10:             PickyModules += {M}
11:         end if
12:         D_current+1 += getResults(M_inv, getInput(D_current))
13:     end for
14:     if PickyModules ≠ ∅ then
15:         breakall
16:     end if
17:     D_current = D_current+1
18: end for
```

4 DETERMINING THE OUTPUT OF THE INVERSE MODULE

The central operations, which are repeatedly performed in the above algorithm, are the test of acceptance of data items by the inverse of a given module `M`, and the calculation of the results (data items) returned by the invocation of the inverse module.

To assess whether a module M_{inv} accepts a given data item `d`, we need to invoke M_{inv} using `d`. If the invocation of M_{inv} terminates successfully, then we can conclude that M_{inv} accepts `d`. Otherwise, if the execution of M_{inv} raises an illegal input exception, then we can conclude that M_{inv} does not accept `d`.

Unfortunately, we cannot perform this test, because we do not have access to M_{inv}. To address this issue, we harvest the (probably) biggest source of information, namely the Web. There are several proposals in the literature that attempted to extract relational data from tables on the Web (see e.g., [3, 11]). Our objective is, however, different. We do not seek to transform all tabular HTML data into a structured format, but instead, identify the input data items that may be candidate for producing a giving data item when used to feed a given module.

We adopt for this purpose a process, composed of three steps: i)- identifying candidate pages, ii)- extracting candidate input data items, and iii)- examining the candidate input data items.

4.1 Identifying Candidate Web Pages

To identify the web pages of interest, we make use of semantic annotations describing the input and output parameters of the module `M`. Indeed, a number of scientific modules are annotated with ontological concepts informing the semantic domain of the input and output parameters of the module, see e.g., bio-tools[2]. We make use of semantic annotations, because often the names of the input and output parameters are non-informative and take values like `in` or `out`. Semantic annotations provide a crisper description of parameter types.

Consider for example that `M` has an input with a semantic domain c_{in} and an output of a semantic domain c_{out}. We issue a query with the following keywords: $\{c_{in}, c_{out}, d\}$ against a Web search engine. We are interested in finding the web pages W_c that contains all of the keywords. We call W_c candidate web pages. If the set W_c is empty, then M_{inv} is likely not to accept the data item `d`. Note that we say *likely*. This is partly because the Web, albeit a large data source, there is no guarantee of it being complete, and partly because the keywords used may fail to locate a Web page that contains the desired information, even if such a page exists.

4.2 Extracting Candidate Input Data Items

For each web page in W_c, we apply information extraction algorithms to retrieve data items that have c_{out} as a semantic domain. The problem we face here is that Web pages are unstructured. Solutions that have been proposed for information extraction, e.g., [3, 7, 11] can be used for this purpose.

As well as the above proposals, in the context of our work, we use recognizers. Indeed, many of the semantic domains, e.g., pathways, enzymes, proteins, etc., in the scientific fields are associated with recognizers, able to identify the semantic domain of a given raw text/string. This applies particularly to accessions, which can be seen as scientific identifiers for entities such proteins, RNAs, etc., as well as other more complex structures such as sequence entries, e.g;, Fasta format, IPR entry, etc. Therefore, if c_{out} is associated with a recognizer, then we apply the recognizer to each of the web pages in W_c. This will result in a list of candidate data items D^c_{in}.

4.3 Examining Candidate Input Data Items

We use the data items in D^c_{in} to feed the execution of the module `M`. It an execution that takes an input data item d_{in} yields a successful execution of `M` and produces as a result `d`, then we can conclude that the inverse M_{inv} accepts `d` and it delivers as a result d_{in}. Note that in the general case, multiple input data items can be associated with `d`. For the purpose of this work, however, we assume that the module `M` and its inverse are functions.

If, on the other hand, none of the candidates in D^c_{in} yields `d` when invoking `M`, then we conclude that M_{inv} is likely not to accept `d`.

[2] https://bio.tools

5 FEASIBILITY STUDY

The approach we have just described raises the following question. *Is the algorithm proposed able to identify the reason why a given data item does not appear in the workflow results?* More specifically, *How effective is our solution in identifying picky modules and missing input data items?*

To answer the above questions, we run a feasibility experiment, in which we used a sample of 6 real-world workflows from the myExperiment repository [3]. We selected workflows that involve deterministic modules, which mean modules that deliver the same result (if any) given the same input. We did not consider workflows that include modules performing data mining operations, for instance. We have also selected workflows for which the inverse modules are also deterministic functions.

5.1 Set-up

We have executed each workflow using example data inputs provided by the workflow authors. In doing so, we used the Taverna workflow systems [12]. We then specified two kinds of queries for each workflow:

- Instance-based why-not query. To assess the ability of the algorithm in answering this type of queries, we randomly selected an output data item d that was returned by the workflow executions. Next, we used our algorithm to see if it is able to reconstruct the lineage of d by harvesting the web to identify the input data items that were responsible for its derivation. We then compared the lineage reconstructed for d with the lineage available in the provenance of the workflow executions previously recorded by the workflow system. This allows us to see if our algorithm was able to successfully identify the input data items responsible for the derivation of d.
- Module-based why-not query This kind of query is used to assess if the algorithm is able to identify picky modules. That is modules that are responsible for the non deliverance of a given data item d by the workflow executions. To do so, for each workflow, we identified a data item that we know it cannot be delivered by the workflow because of a given (picky) module. We then used our algorithm to assess if it is able to identify such a module.

In total we had 6 queries of the first kind, which we denote by $\{q_1^+, \ldots, q_6^+\}$], and 6 queries of the second kind, which we denote by $\{q_1^-, \ldots, q_6^-\}$. The + sign indicates that we should be able to reconstruct the provenance of the why-not query up to the workflow input, and the − sign indicates that we should not be able to do so, and instead identify the picky module.

5.2 Results

Of the queries $\{q_1^+, \ldots, q_6^+\}$, our algorithm was able to successfully constructs the provenance of the why-not query up to the workflow input for 3 queries. Most of the modules composing these workflows, namely 8 out of 11, provides information about the input and output datasets on the Web using Tabular formats.

After examination of the three remaining workflows, we found that one them utilizes proprietary data sources, the content of which is not accessible on the surface web. The last two workflows, on the other hand, contain modules that manipulate excerpt from HTML web pages. Because of this, our algorithm was not able to find the content on the Web of the input and output of those modules. However, we made a key observation thanks to these two workflows that will allow us to improve the quality of the results delivered by our algorithm. Indeed, these modules that were problematic for our algorithm perform format transformation, which we refer to as Shims in scientific workflows. Such modules could be ignored (skipped) by our algorithm. For example, once ignored, our algorithm was able to identify the inputs data items of the remaining modules in the workflows.

We also measured the number of Top-k web pages that needed to be examined to identify the input data item corresponding to a given output data item. On average, we needed to examine the content of the 4 top web pages returned by the key-word search engine[4]. In several cases, however, the top web page was the right one, in the sense that it contained the input data item we are after.

Regarding the queries $\{q_1^-, \ldots, q_6^-\}$, our algorithm was more successful in the sense that it was able to correctly identify 4 picky modules out of 6. For two remaining workflows, the module that was identified as picky by our algorithm was not the correct one. After examination, it transpired that for certain modules the corresponding data item could not be found on the web. Again this issue was due to shims modules the input and output data items are not published on the Web.

To sum up, this small feasibility study has shown that our method is promising. It has also brought some insights into the way our solution can be improved. Our ongoing work includes: i)- tuning our algorithm to deal with shims modules in a workflow, ii)- explore new source of information for identifying picky modules, iii)- extending our solution to cases where the inverse of a module is not a function, and iv)- an experiment involving a large number of scientific workflows.

REFERENCES

[1] S. S. Bhowmick, A. Sun, and B. Q. Truong. 2013. Why not, WINE?: towards answering why-not questions in social image search. In *ACM Multimedia Conference, MM '13, Barcelona, Spain, October 21-25, 2013*. ACM, 917–926. https://doi.org/10.1145/2502081.2502098

[2] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. 2014. Query-Based Why-Not Provenance with NedExplain. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*. OpenProceedings.org, 145–156.

[3] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *PVLDB* 1, 1 (2008), 538–549. http://www.vldb.org/pvldb/1/1453916.pdf

[4] Adriane Chapman and H. V. Jagadish. 2009. Why not?. In *Proceedings of SIGMOD*. ACM, 523–534.

[5] L. Chen, J. Xu, X. Lin, C. S. Jensen, and H. Hu. 2016. Answering why-not spatial keyword top-k queries via keyword adaption. In *ICDE*. IEEE-CS, 697–708. https://doi.org/10.1109/ICDE.2016.7498282

[6] Rafael Ferreira da Silva, Ewa Deelman, Rosa Filgueira, et al. 2016. Automating environmental computing applications with scientific workflows. In *Proceedings of eScience*. IEEE, 400–406.

[7] Maeda F. Hanafi, Azza Abouzied, Laura Chiticariu, and Yunyao Li. 2017. SEER: Auto-Generating Information Extraction Rules from User-Specified Examples. In *Proceedings of the CHI Conference*. ACM, 6672–6682.

[8] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A survey on provenance: What for? What form? What from? *VLDB J.* 26, 6 (2017), 881–906.

[9] Melanie Herschel and Mauricio A. Hernández. 2010. Explaining Missing Answers to SPJUA Queries. *PVLDB* 3, 1 (2010), 185–196.

[10] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2008. On the provenance of non-answers to queries over extracted data. *PVLDB* 1, 1 (2008), 736–747.

[11] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web using Column Keywords. *PVLDB* 5, 10 (2012), 908–919.

[12] K. Wolstencroft, R. Haines, D. Fellows, et al. 2013. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research* (2013), W557–W561.

[3] www.myexperiment.org

[4] We used the Google search engine for our experiment.