Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow*

João Felipe Pimentel[1], Saumen Dey[2], Timothy McPhillips[3], Khalid Belhajjame[4], David Koop[5], Leonardo Murta[1], Vanessa Braganholo[1], Bertram Ludäscher[3]

[1] Universidade Federal Fluminense, Brazil
[2] University of California at Davis, USA
[3] University of Illinois at Urbana-Champaign, USA
[4] Université Paris-Dauphine, France
[5] University of Massachusetts Dartmouth, USA
{jpimentel, leomurta, vanessa}@ic.uff.br, scdey@ucdavis.edu, tmcphillips@absoluteflow.org, khalid.belhajjame@dauphine.fr, dkoop@umassd.edu, ludaesch@illinois.edu

Abstract. The noWorkflow and YesWorkflow toolkits both enable researchers to capture, store, query, and visualize the provenance of results produced by scripts that process scientific data. noWorkflow captures prospective provenance representing the program structure of Python scripts, and retrospective provenance representing key events observed during script execution. YesWorkflow captures prospective provenance declared by annotations in the comments in scripts, and supports key retrospective provenance queries by observing what files were used or produced by the script. We demonstrate how combining complementary information gathered by noWorkflow and YesWorkflow enables provenance queries and data lineage visualizations neither tool can provide on its own.

1 Introduction

Methods for harvesting provenance information from scripts and runs of scripts have been of great recent interest to the provenance research community, and the resulting tools have received increasing attention from users of scripting languages in the natural sciences. Some of these approaches are language-specific, e.g., noWorkflow[6] [4] (Python) and RDataTracker [2] (R scripts), while others are language-independent, e.g., YesWorkflow[7] [3] and LLVM/SPADE [5]. Using such tools often entails annotating the scripts [2,3], monitoring executing scripts as they run [5,4], or both.

Approaches that do not require annotation, such as noWorkflow (NW), rely on the structure of the code itself to build prospective and retrospective provenance graphs. NW includes the actual function and variable names in the prospective provenance

* Extended Technical Report Version
[6] For "**n**ot **o**nly workflow", emphasizing that scripts need provenance tracking, too.
[7] Which can be read as "Yes, scripts can be workflows, too!"

records, and it depends on records of run-time function calls to capture the retrospective provenance of script outputs. Consequently, the less meaningful variable and function names are in a script, the less clear the provenance query results and visualizations will be to scientists using the script. noWorkflow thus excels where Python programs are engineered for maintainablity, testability, code reuse, and long-term user support.

YesWorkflow (YW) is an example of a tool that largely ignores the code portions of a script, and instead depends on script authors (or users) adding annotations via comments in scripts. YW annotations declare the scientifically significant steps implemented by code blocks in a script, and the routes of dataflow between these steps. Annotations optionally assign meaningful names to actual (often obscurely named) code-level entities. Consequently, YesWorkflow users need not rename variables, move code into functions, or otherwise refactor scripts that already have been used to produce results (research transparency requires disclosure of the scripts actually used). YW users can capture provenance from a working script without incurring the regression testing costs that refactoring entails. YW thus provides benefits even when scripts are written rapidly in the course of competitive, time-critical research, and when researchers employ scripts that they do not intend to maintain further or to distribute and support.

Given the contrasting aims of noWorkflow and YesWorkflow and the differences in the approaches they take, it is not surprising that each supports queries and visualizations that the other cannot support on its own [1]. We show that there are provenance artifacts of great interest to researchers that only a combination of YW and NW provenance can produce. Achieving this combination requires mapping between common entities in both provenance models, and jointly querying the provenance information represented by each system. We refer to the joint provenance model, the system-spanning queries, and the resulting visualizations collectively as YW*NW.

2 NW and YW Example Queries

We use the Python script described by McPhillips et al. [3] to demonstrate the kinds of provenance queries noWorkflow, YesWorkflow, and the combination of both support. This script simulates acquisition of diffraction images during macromolecular X-ray crystallography experiments involving multiple samples. The script reads previously measured data quality statistics for each sample from an input spreadsheet; rejects samples that do not meet a minimum quality criterion; and for each accepted sample produces raw and corrected diffraction images according to a data collection strategy that depends on properties of the samples. Although the script only simulates data collection[8], the order of task execution, the sequence of data production events, and the resulting pattern of dependencies between input, intermediate, and final data items closely mimic those of a real experiment. Queries that probe these dependencies are therefore illustrative of meaningful uses for provenance information. The complete script, marked up with YW annotations, is available on GitHub [7], and a more complete explanation is provided by McPhillips et al. [3].

[8] The data acquistion and processing steps implemented as functions in the example script produce meaningless data, and real-world implementations [6] typically delegate these tasks to automated instruments or other programs.

noWorkflow. Examples of **prospective** provenance queries of this script that NW supports include: *What functions does the top-level function call? Are any functions defined in the script not called by the top-level function?*

NW can answer **retrospective** provenance queries about runs of this script, such as: *What values did the variable* rejected_sample *take during writes to files referred to by the* rejection_log *variable? What files were written during calls to the* transform_image *function? How many files were written while the* accepted_sample *variable had the value* DRT240*? What variables carry values returned by the* calculate_strategy *function to calls to the* collect_next_image *function? What parameters to the top-level function can effect the results returned by calls to* calculate_strategy*?*

NW also can answer queries about the execution context: *Which user executed the script? What version of Python was used?*

YesWorkflow. YW provenance queries refer to annotated code blocks (workflow steps) rather than to Python functions, and to data names declared via YW annotations instead of to Python variables. Queries of **prospective** provenance supported by YW include: *What are the names of steps that comprise the top-level workflow implemented by the script? What data is output by the* collect_data_set *step? What code blocks provide input directly to that step? What data is* corrected_image *directly derived from?*

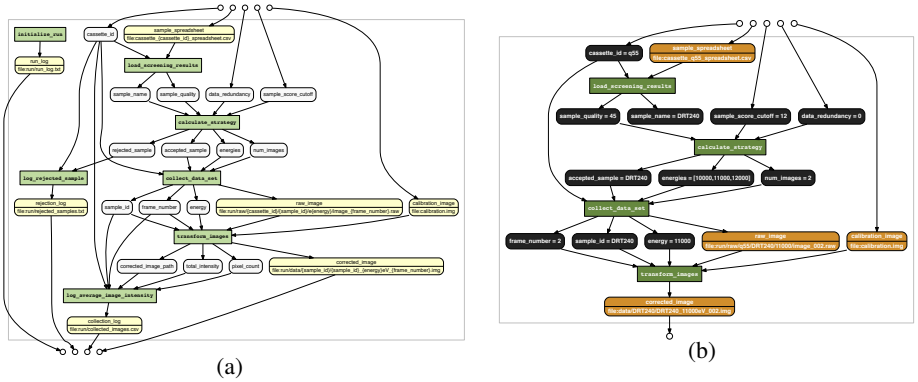
YW can answer **retrospective** provenance queries [3], including: *What samples did the run of the script collect images from? What energies were used during collection of images from sample* DRT240*? Where is the raw image from which corrected image* run/data/DRT322/DRT322_10000eV_001.img *is derived? Are there any raw images for which there are* ***no*** *corresponding corrected images?*

3 Querying the Combined YW*NW Provenance

Queries that must be answered by combining NW and YW provenance generally involve references both to Python functions or variables *and* to code blocks or data declared via YesWorkflow annotations. Examples include: *Can the* sample_id *output of the* collect_data_set *step ever produce values other than those provided via the* accepted_sample *input to this step? What Python functions may be called as part of the* calculate_strategy *step? What was the set of* energies *produced by the* compute_strategy *step for sample* DRT322?

As these queries demonstrate, the combination of NW and YW provenance enables code-level entities such as Python functions and variables to be queried in terms of data and workflow steps meaningful to the user (and vice versa). Such queries are useful for understanding runs of the script in ways that neither NW nor YW enable on their own. Generalizing these queries yield meaningful visualizations of the full lineage of any product of the script.

Figure 1(a) depicts the YW-generated prospective provenance graph of the demonstration script. The yellow, rounded boxes in the graph correspond to data elements named as inputs and outputs to YW-annotated code blocks in the script. The code blocks involved in producing the diffraction image in question are depicted in the diagram as green rectangles.The figure does not show the names of Python variables involved or the names of Python functions called while producing the image. Only the YW-declared



(a) (b)

Fig. 1. Combining complementary provenance: (a) YW-generated workflow graph (prospective provenance). (b) Hybrid of YW prospective provenance and NW retrospective provenance; nodes and edges comprise the subgraph of the YW model of the script upstream of a single corrected_image; values in nodes are extracted from the NW runtime records of corresponding variable values leading to a particular image. See Appendix B for larger versions of these figures.

names for workflow steps and data are used. Values of unannotated Python variables are fully hidden from view. The result is a much simplified rendering of data lineage that is nonetheless complete from a scientific point of view. NW provenance on its own can be used to render a complete graph of variable values and function invocations leading to a particular output. However, the size and complexity of such graphs can be overwhelming (Fig. 3), and insights these graphs provide often are limited by obscure names given to variables and functions in many scripts.

Joint YW*NW queries and visualizations yield novel insights of great value to scientists who employ scripts in their research. The combination of noWorkflow and YesWorkflow achieves this without requiring the major adaptations to code often needed to run existing software in a scientific workflow management system. Indeed, YW*NW provides many benefits of provenance management without requiring working code to be refactored at all.

4 Demonstration

In our demonstration we will highlight the benefits of harvesting, querying, and visualizing provenace with noWorkflow in conjunction with YesWorkflow. Starting with a directory containing just the example script and input files, we will (1) highlight how YW annotations can be visualized as prospective provenance using YesWorkflow (Fig. 1(a)); (2) run the script using noWorkflow and relate the resulting data file names and locations to the YW prospective provenance; (3) query the script and its outputs using noWorkflow and YesWorkflow separately to illustrate what each tool can do on its own; and (4) execute joint YW*NW queries that determine the lineage of a single data product and produce a visualization analogous to Fig. 1(b).

A companion GitHub repository for this demonstration is available [7]. It includes the data collection simulation script discussed in this paper; the files produced by a run of this script; the provenance information produced by noWorkflow and YesWorkflow; and helper scripts for running the queries mentioned above and for producing Figure 1. noWorkflow and YesWorkflow both are available on GitHub and can easily be installed.

References

1. Dey, S., Belhajjame, K., Koop, D., Raul, M., Ludäscher, B.: Linking Prospective and Retrospective Provenance in Scripts. In: Theory and Practice of Provenance (TaPP). Edinburgh, Scotland (2015)
2. Lerner, B., Boose, E.: RDataTracker: Collecting Provenance in an Interactive Scripting Environment. In: Theory and Practice of Provenance (TaPP). Cologne, Germany (2014)
3. McPhillips, T., Bowers, S., Belhajjame, K., Ludäscher, B.: Retrospective Provenance Without a Runtime Provenance Recorder. In: Theory and Practice of Provenance (TaPP). Edinburgh, Scotland (2015)
4. Murta, L., Braganholo, V., Chirigati, F., Koop, D., Freire, J.: noWorkflow: Capturing and Analyzing Provenance of Scripts. In: International Provenance and Annotation Workshop (IPAW). pp. 71–83. Cologne, Germany (2014)
5. Tariq, D., Ali, M., Gehani, A.: Towards Automated Collection of Application-level Data Provenance. In: Theory and Practice of Provenance (TaPP) (2012)
6. Tsai, Y., McPhillips, S.E., González, A., McPhillips, T.M., Zinn, D., Cohen, A.E., Feese, M.D., Bushnell, D., Tiefenbrunn, T., Stout, C., Ludäscher, B., Hedman, B., Hodgson, K.O., Soltis, S.M.: AutoDrug: fully automated macromolecular crystallography workflows for fragment-based drug discovery. Acta Crystallographica Section D: Biological Crystallography 69(5), 796–803 (2013)
7. Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow. https://github.com/gems-uff/yin-yang-demo/

A Demonstration Script (Walk-Through)

The following are the current instructions (subject to change) for the demonstrator of the prototype. The actual instructions will be available from the demonstration site [7].

1. Start by showing a directory containing the simulate_data_collect script, the two input files it needs, and a yw.properties file.
2. Briefly show the Python script, highlighting the YW annotations for a couple of the workflow steps to illustrate how the steps are named, how there are @in ports that match @out ports on other steps, how the arguments to @in and @out name actual variables in the script, and how some ports are qualified with @uri annotations.
3. Run yw graph to produce the prospective provenance as a DOT (Graphviz) file, and walk through the workflow automated by the script, noting the blocks and ports that were mentioned in 2.
4. Open the extractfacts.P and modelfacts.P files created by YW. Explain some of the facts that will be most relevant to YW*NW integration.
5. Run a YW model (prospective) query or two and confirm that the answers make sense.
6. Run the script using noWorkflow.
7. Use the tree command on the newly created run directory to show what files were created. Show how the files created correspond to @uri annotations in the script.
8. Run now show -f to show files collected by NW, and which activations opened them.
9. Run now export -r > kb.pl to export NW provenance to prolog.
10. Run a NW prospective query or two and confirm that the answers make sense. Point out that these queries refer to different kinds of things than the YW queries (functions and variables rather than steps, channels, etc.)
11. Run yw recon, open reconfacts.P, and explain a few facts.
12. Run a couple retrospective queries using YW.
13. Run a couple retrospective queries using NW.
14. Export NW dataflow.
15. Run the YW*NW integration script, producing a nw_yw_facts.P file and explain a few of the (new) facts.
16. Ask the audience which of the images (raw or corrected) in the run directory tree they would like to see the full lineage of. Run a script that renders the image's lineage as a DOT file.
17. Display and walk through the YW*NW lineage of the image and compare to the YW prospective view of the script.
18. Run a script that produces the NW-only lineage and compare to the YW*NW image. Highlight advantages of the latter for users of the script who haven't read the script.
19. Recall/emphasize the benefits of automation (say: "Look ma, no hands!" ;-)

B Provenance Figures (Close-Up)

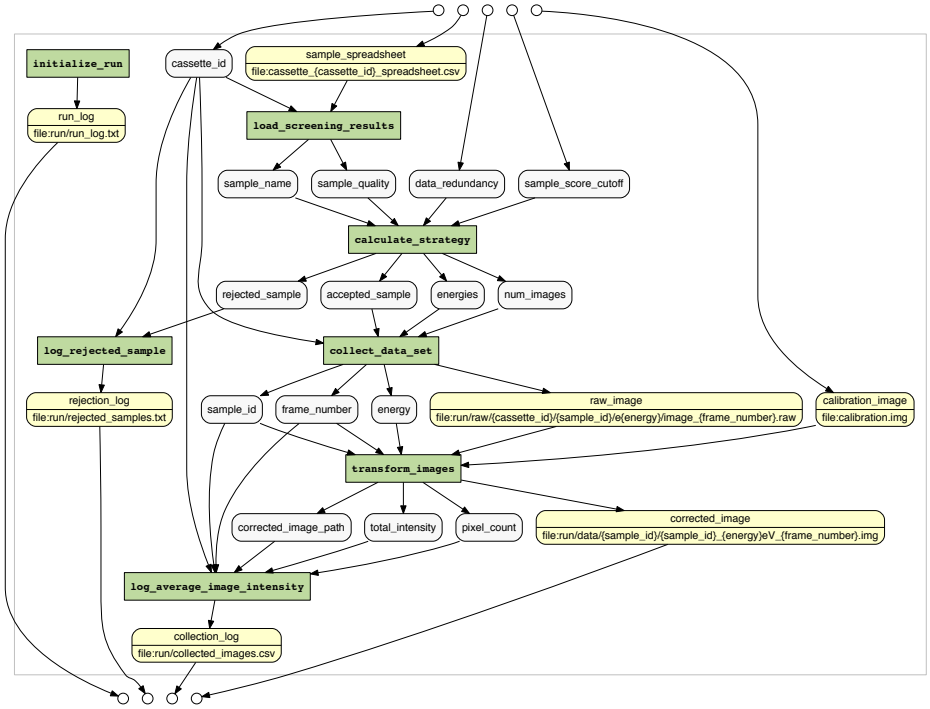


Fig. 2. YW-generated prospective provenance graph (workflow) for the demonstration script simulate_data_collection (large version of Fig. 1(a)).

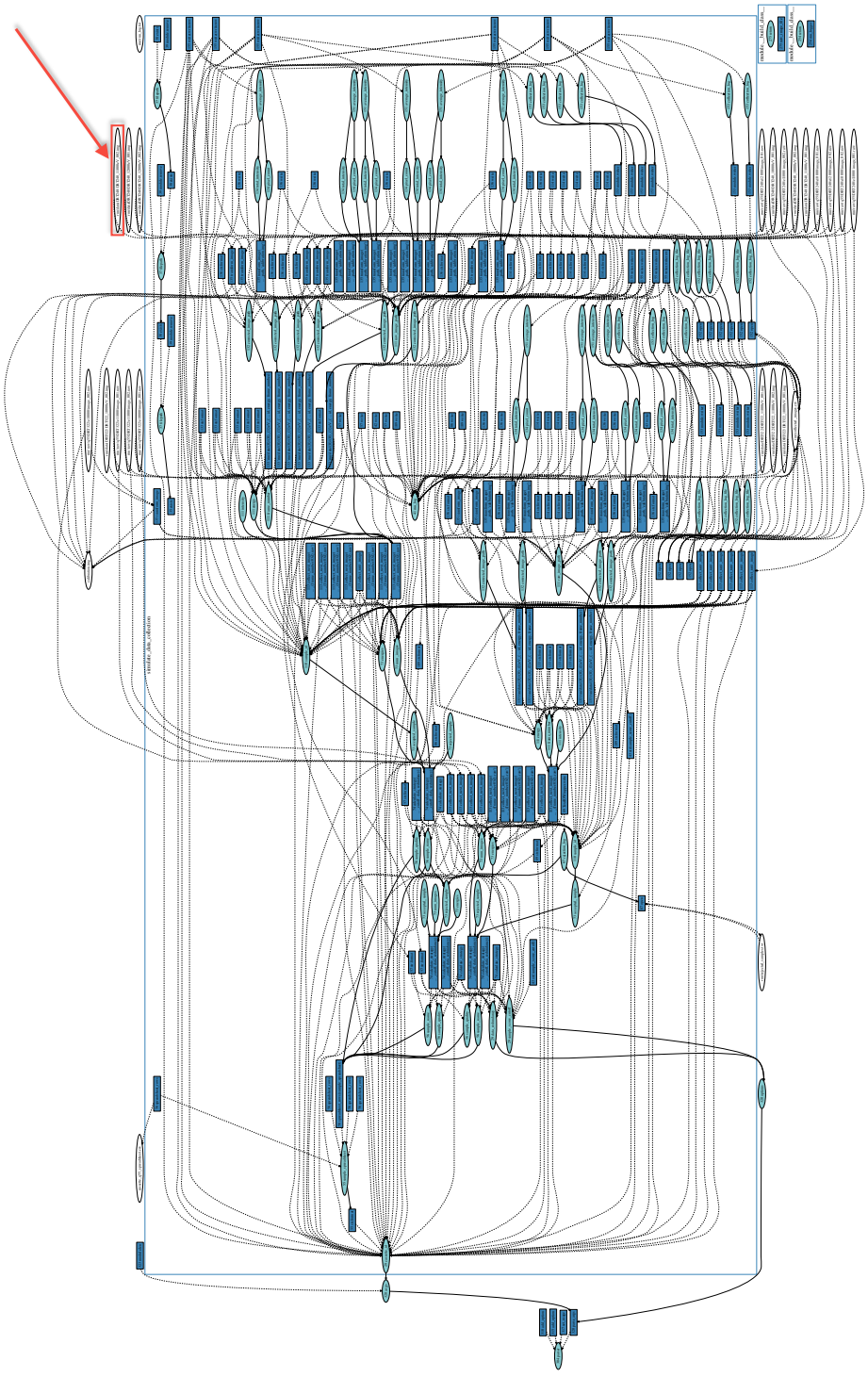


Fig. 3. NW-generated retrospective provenance graph for the demonstration script. Not meant to be readable (unless viewed on-screen), but to give an overview. The arrow points at the result image file of interest, i.e., the output file whose lineage is depicted in Figure 1(b) (and Figure 4).

Fig. 4. Result of combining complementary YW and NW provenance (large version of Fig. 1(b)): Only the workflow subgraph upstream of corrected_image in Fig. 2 is relevant provenance for the production of a corrected image. This YW subgraph is combined with NW-recorded provenance (Fig. 3) to fill in the missing attribute values in the subgraph with the recorded values of variables from the runtime trace.