P+RProv: Prospective+Retrospective Provenance Graphs of Python Scripts

Vitor Gama Lemos, João Felipe Pimentel
Bruno Erbisti, Vanessa Braganholo

Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brazil
{vitorlemos,jpimentel,berbisti,vanessa}@ic.uff.br

Abstract. The evolution of technology has enabled scientists to advance the automation of scientific experiments. Many programming languages have become popular in the scientific environment, especially scripting languages, due to their high abstraction level and simplicity, allowing the specification of complex tasks in fewer steps than traditional programming languages. Due to these features, lots of scientists model their scientific experiments in scripting languages to ensure data management and results control. However, this type of experiment usually generates large volumes of data, making data analysis and threat mitigation difficult. To fill in this gap, we propose P+RProv, an approach to aid scientists in understanding the structure of Python scripts and their results.

Categories and Subject Descriptors: [**Database Management Systems**]: Data provenance; [**Software and its Engineering**]: Flowcharts

Keywords: diagrams, prospective provenance, provenance visualization, scripts

1. INTRODUCTION

Scientists use scripts as a strategy to ensure control and management during the execution of experiments, consequently speeding up the seek of discoveries. Due to technological evolution, this is becoming increasingly common. Scripting languages are widely used because of their simplicity, allowing the specification of complex tasks in fewer steps than traditional programming languages [Pimentel et al. 2019]. Even with these benefits, evaluating and manipulating scientific experiments in a computing environment can become extremely complex because of the large volumes of data that need to be analyzed. For this reason, provenance gained relevance as a solution to help scientists interpret and understand the details of their experiments [Freire et al. 2008; Herschel et al. 2017; Pimentel et al. 2019; Simmhan et al. 2005].

In brief, provenance refers to any information describing the production process of an end product, which can be anything from a piece of digital data to a physical object [Herschel et al. 2017]. For scripts, it can be used for multiple purposes, such as to identify dependencies on scripting codes, collect data about the system infrastructure (operating system version, environment variables, libraries, and hardware components), and obtain information of the script execution (execution time, function activations, input data, and output data). Moreover, provenance can also be useful to allow developers to optimize their scripts and eliminate possible inconsistencies in the programming code [Linhares et al. 2019].

Because of the challenges related to the execution of experiments in a computational environment, various approaches have been proposed for collecting, managing, and analyzing the provenance of scripts, such as noWorkflow [Murta et al. 2014], YesWorkflow [McPhillips et al. 2015], ProvenanceCu-

rious [Huq et al. 2013], Sumatra [Davison 2012], Visustin [Oy 1997] and RDataTracker [Lerner and Boose 2014]. These tools are essential to help scientists maintain the highest quality and reliability of the results of their experiments. Moreover, there is a keen interest in using visual models to represent provenance and abstracting complex data. Often the script provenance is visually represented by directed graphs, process diagrams, or algorithm flowcharts [Pimentel et al. 2019].

However, many existing approaches do not connect prospective and retrospective provenance in an intuitive format. Prospective provenance refers to data related to the script structure, as variables, loops, functions, and conditional statements. Retrospective provenance refers to data collected over an execution cycle, including information on the performed processes, the function activations that occurred in the code, and the variable values [Freire et al. 2008]. Prospective provenance graphs produced by existing tools are hard to understand, especially for people who have less contact with programming languages. Besides, even approaches that produce more intuitive diagrams, such as Visustin [Oy 1997], do not guarantee the script code is correct since there is no syntax validation of the code that is used to generate the graphs.

To fill in this gap, we propose P+RProv, a prospective+retrospective provenance diagram generator that uses provenance collected during the execution of the script. This way, we can guarantee that the generated diagrams correspond to a correctly structured script since it was executed without errors. Furthermore, P+RProv associates data from the execution of the script, which allows joining the two types of provenance (prospective and retrospective) in a single diagram. Combining prospective and retrospective provenance helps scientists understand the script's structure and the results produced by them. In the scientific field, understanding the results of the experiments and at the same time understanding which part of the code is producing these results is fundamental because it optimizes the time of the scientists in the analysis of results and facilitates the correction of eventual problems such as points of failure, logic errors, and inconsistent structures.

More specifically, our approach aims at supporting scientists in understanding the structure of Python experiments using code diagrams. Python is widespread in the scientific environment and widely used in the design of computational scientific experiments [Pimentel 2021]. To increase the abstraction level, we use diagrams based on algorithm flowcharts, a popular notation in the academic community.

To evaluate the efficiency and effectiveness of P+RProv, we conduct three experimental evaluations. The first one compares P+RProv with two other related approaches: Visustin [Oy 1997] (which uses flowcharts to display prospective provenance) and ProvenanceCurious [Huq et al. 2013]. In this evaluation, we only selected the tools dedicated to generating diagrams based on the structure of Python script – such tools consider the prospective provenance. For this, we did not consider tools that do not automatically generate diagrams, such as YesWorkflow, which requires users to create annotations in scripts. Due to these criteria, other tools were not used for this evaluation, such as RDataTracker [Lerner and Boose 2014], and Sumatra [Davison 2012]. The results of our evaluation indicate that P+RProv is as effective as Visustin, and as efficient as Visustin in complex tasks. However, Visustin does not link prospective and retrospective provenance, and does not guarantee that the structure of the script is correct (since the diagram is not generated from execution data as it is in P+RProv). Also, P+RProv achieved far superior results in effectiveness and efficiency when compared to ProvenanceCurious, an approach that uses provenance notations to represent the prospective provenance of scripts.

The second evaluation compares P+RProv with scripts alone. Again, there were no statistical differences in the results. However, users that are not from the Computer Science field felt more comfortable with P+RProv than with using the scripts alone. This indicates there is room for further investigation with a larger group of non Computer Science users. The results of our experimental evaluation are essential to understand the limitations of our proposal and improve P+RProv visualization methods [Weintraub 2016].

Finally, the third evaluation compares the pure prospective provenance diagrams of P+RProv with the combined provenance graphs (union between prospective and retrospective provenance), also generated by P+RProv. In this case, there was a statistically significant result in one of the more complex tasks in the efficiency evaluation. Moreover, most participants responded that they prefer to use the combined provenance diagrams.

We organize this paper as follows: In Section 2, we present the related work. Section 3 describes our proposed approach. Section 4 provides an evaluation of our approach and discusses the results. Finally, Section 5 provides conclusions and future work.

2. RELATED WORK

Because of the challenges related to the execution of experiments in a computational environment, various approaches have been proposed for collecting, managing, and analyzing the provenance of scripts, such as noWorkflow [Murta et al. 2014], YesWorkflow[McPhillips et al. 2015], ProvenanceCurious [Huq et al. 2013], Sumatra [Davison 2012], Visustin [Oy 1997] and RDataTracker [Lerner and Boose 2014]. These tools help scientists maintain the highest quality and reliability of the results of their experiments. Besides, many of these tools use visual models to represent the source of scripts and alleviate the complexity of programming and understanding the script code.

noWorkflow[Murta et al. 2014] is an open-source tool that systematically and transparently collects the provenance of scripts. To do this, noWorkflow collects and stores data during the script execution. noWorkflow can use this data to represent retrospective provenance through directed graphs. Although noWorkflow collects prospective provenance from scripts, noWorkflow does not build or generate prospective provenance graphs. P+RProv fills in this gap, consuming provenance data collected by noWorkflow to create diagrams based on the code structure, and it also leverages retrospective data to generate combined diagrams.

ProvenanceCurious [Huq 2013; Huq et al. 2013] is a tool that collects prospective provenance of Python scripts. It allows scientists to view a graphical representation of Python scripts. ProvenanceCurious helps scientists not only in interpreting a script but also in visualizing the data dependency relationships between operations.

In the context of tools to collect provenance from scripts, we can cite RDataTracker [Lerner and Boose 2014], a library designed for collecting data provenance during the execution of R scripts and console sessions. The R language is commonly used to create experiments focused on mathematics and statistics. Thus, RDataTracker has resources to examine and visualize the source data of scripts written in R, maintaining a high level of abstraction to keep the provenance graphs comprehensible to users [Lerner and Boose 2014]. R is a suitable language for manipulating data and applications related to statistical studies, while Python is a more versatile and widely used language [Ozgur et al. 2017].

YesWorkflow [McPhillips et al. 2015] collects and represents prospective provenance in script-based experiments. It can generate prospective provenance graphs without executing the code. YesWorkflow uses annotations in the script to be able to generate the provenance diagrams. These annotations are guidelines to generate directional graphs in YesWorkflow, where the vertices represent blocks of code and the edges represent the data flow (input and output). This approach is advantageous to syntax independence, as it allows the same annotations to be incorporated in other programming languages. In contrast, the annotation model does not guarantee the integrity of annotations, which may quickly make annotations obsolete. Another drawback is that scientists must create their annotations in the script code before generating the provenance diagrams.

Visustin [Oy 1997] is a flowchart generator system for software developers. It is not limited to just the Python language, generating flowchart diagrams of other programming languages. Besides,

Visustin automatically converts source codes into UML activity diagrams. Visustin diagrams make it possible to view the code structure and comments. Although it is a useful tool for representing Python code as diagrams, it does not allow users to check for code correctness. In fact, Visustin does not execute the codes and does not guarantee that they are syntactically correct. The generated diagrams are based only on basic rules of the source code language and indentation.

None of the existing approaches connect prospective provenance with retrospective provenance. To fill in this gap, in the next section we propose P+RProv.

3. P+RPROV

3.1 Overview

P+RProv consumes provenance data collected by noWorkflow [Murta et al. 2014; Pimentel et al. 2017] to generate a combined provenance diagram referring to both the prospective and the retrospective provenance of Python scripts. It uses noWorkflow to collect prospective provenance (which in noWorkflow is called *definition* provenance) and the Graphviz framework [Ellson et al. 2004] to generate a provenance diagram based on flowchart. We opted for using noWorkflow as it collects both prospective and retrospective provenance when running a Python script. This makes P+RProv an automatic tool. That is, it is not necessary to re-execute the script or manually insert annotations for it to generate the diagrams of an experiment. Thus, P+RProv can represent the entire structure of a script written in Python 3, recognizing various structures of this programming language (which we call code blocks), such as conditional statements, loops, variables, calls, functions, imports, and other identifiers.

3.2 Provenance Visualization

P+RProv represents the provenance data using diagrams. By default, P+RProv generates visual representations based on the prospective provenance (code structure). Optionally, P+RProv allows the generation of a combination of the prospective and the retrospective provenance. The prospective provenance diagrams contain components of Python code, such as conditional structures, functions, function calls, repetition structures, and variables. When this diagram includes retrospective provenance, P+RProv indicates in the diagram the structures that were activated (executed) and also the content (value) of the variables in a given line of code.

Figure 1 shows the symbols used in P+RProv diagrams for representing prospective and retrospective provenance of Python scripts. For items in the prospective provenance diagram, the cyan rectangles indicate calls, assignments with math operation, and other reserved components (such as else and except statements). The cyan diamond indicates conditional structures, the cyan ellipse indicates a loop, the pink rectangles indicate variable attribution or package dependencies. For items in the retrospective provenance diagram, the white annotation block represents informational items, the yellow annotation block represents variable content, and the light gray annotation block represents components that were not activated during the script execution.

Figure 2 shows a diagram generated by P+RProv, in which only the prospective provenance is represented. This diagram was generated to represent the prospective provenance of the script shown on the right-hand side of the Figure. In this diagram, the cyan rectangles indicate calls, the cyan ellipse indicates a loop, the pink rectangles indicate variable attribution, and the white annotation block indicates a loop condition.

Figure 3 shows a diagram generated by P+RProv to represent the provenance of the script on the right-hand side of the figure. This diagram combines prospective and retrospective provenance. In this figure, the cyan squares indicate function calls, the pink squares indicate increment or attribution

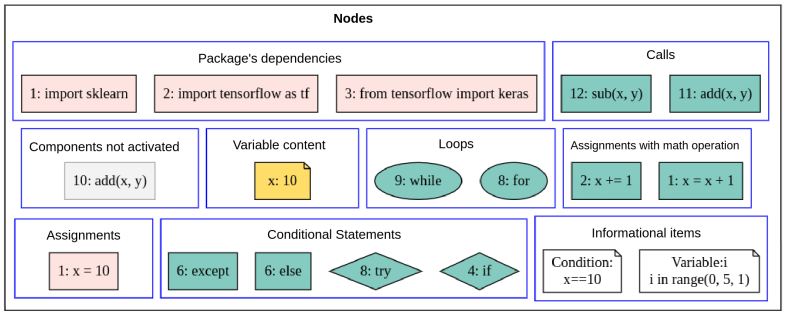
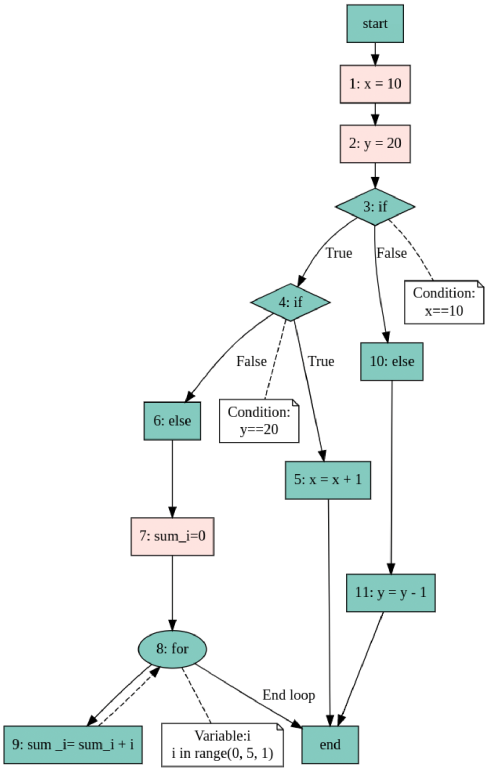


Fig. 1. Symbols (with sample content) used to represent nodes in P+RProv diagrams.



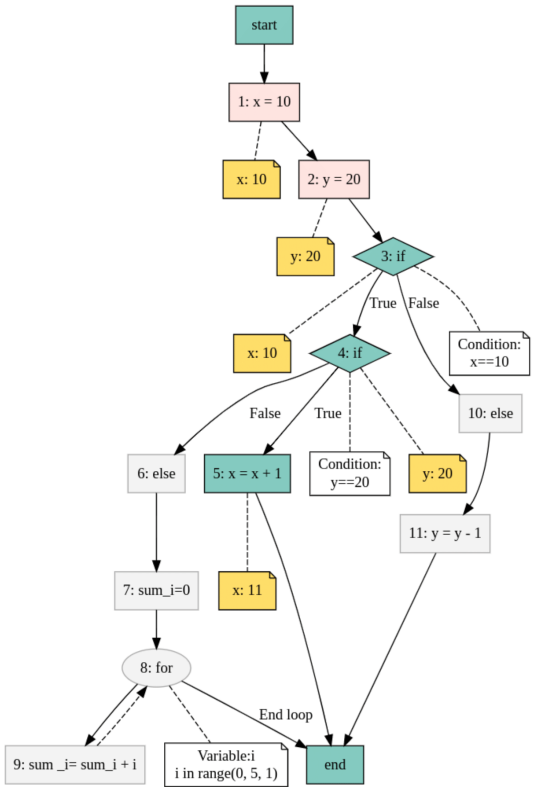

```
x = 10
y = 20
if x == 10:
  if y == 20:
    x = x + 1
  else:
    sum_i = 0
    for i in range(0, 5,
    1):
      sum_i = sum_i + 1
else:
  y = y + 1
```

Fig. 2. P+RProv diagram showing only prospective provenance corresponding to the script on the right hand-side.

of variables, the cyan diamonds indicate conditional structures, the yellow squares indicate the retrospective provenance (state of the variable at that time). Finally, the gray rectangles indicate parts of the script code that have not been executed.

3.3 Architecture

As mentioned before, P+RProv obtains the prospective provenance by accessing provenance data collected by noWorkflow [Murta et al. 2014; Pimentel et al. 2017]. In noWorkflow, prospective provenance is called definition provenance. Definition provenance comprises the script structure, including function definitions, arguments, and function calls [Pimentel et al. 2017]. All code structure data is stored in the noWorkflow SQLite database. For this reason, before starting P+RProv, users need to




```
1  x = 10
2  y = 20
3  if x == 10:
4    if y == 20:
5      x = x + 1
6    else:
7      sum_i = 0
8      for i in range(0, 5,
       1):
9        sum_i = sum_i + 1
10 else:
11   y = y + 1
```

Fig. 3. P+RProv diagram that combines prospective and retrospective provenance. The diagram represents the provenance of the script on the right hand-side.

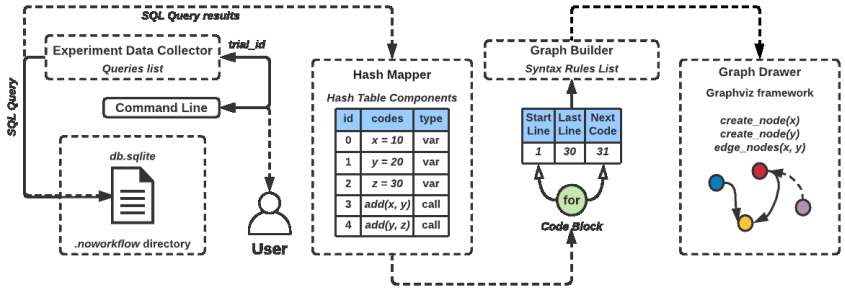


Fig. 4. Architecture of P+RProv.

execute the script on noWorkflow. In noWorkflow, the execution of a given experiment script is called a *trial* [Murta et al. 2014]. We use the trial number to connect P+RProv to experiment data stored by noWorkflow, allowing users to access details about any previously executed trials. To generate a diagram in P+RProv, users need only to enter the *trial id* in the command line interface: `prospective trial <id>`.

We divide the P+RProv architecture into four main modules: *Experiment Data Collector*, *Hash Mapper*, *Graph Builder*, and *Graph Drawer*. Figure 4 presents the architecture.

The *Experiment Data Collector* connects to the noWorkflow database (stored in the *.noWorkflow* directory that is automatically created by noWorkflow inside the directory where the script is stored)

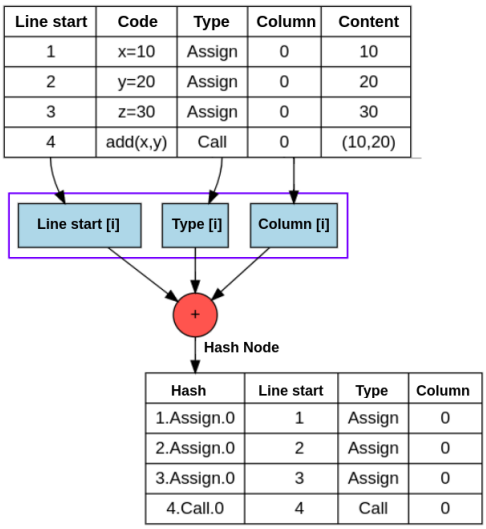


Fig. 5. Mapping of diagram nodes.

and consumes the definition provenance data. These data are useful to create relationships between code blocks. After this process, P+RProv stores the data on a key-value dictionary. The key-value dictionary contains details about each code block component, such as the start line, last line, and code tags (loop, condition, variable, calls, classes).

The *Hash Mapper module* identifies and organizes the diagram's nodes. There is a unique key for each node, allowing P+RProv to identify a node in any part of the diagram. It is possible to use reverse engineering to recover the keys. In this scheme, it is possible to access a specific diagram node at any time, allowing it to connect it to other components or their configuration (name, color, font size, title, subtitle). We use this same heuristic to connect retrospective provenance data to the diagram nodes, such as variable values and function return values. Figure 5 shows how P+RProv creates the code hashes obtained from the provenance data in previous steps. To build the hash, we select the start line number, the code type (which also represents the node's shape and color), and the column number.

Finally, P+RProv performs component analysis to rearrange the code and connect the nodes. There are pre-established rules to indicate how the nodes should be connected. For this reason, the execution time of this module is longer when compared to the other P+RProv modules. Each code block type has a heuristic to build it. As an example, the relationships of a loop are different from the ones of a conditional structure. The *Graph Builder module* establishes the rules for how nodes should be connected (considering the Python 3 syntax rules and flowchart notations [Nassi and Shneiderman 1973]). The Graph Builder is integrated with the *Graph Drawer module*, which is responsible for drawing code diagrams using Graphviz [Ellson et al. 2004], an open-source tool package for drawing graphs in DOT language. Figure 6 shows the nodes being connected using a function from the Graphviz framework, where the nodes hashes that must be connected in the diagram are passed as function parameters.

4. EXPERIMENTAL EVALUATION

To evaluate the efficiency and effectiveness of P+RProv, we performed an online experiment in three phases. In all phases, participants received forms with tasks that involved identifying the script semantics using provenance diagrams. In the first phase, we compare P+RProv with two similar approaches: Visustin [Oy 1997] and ProvenanceCurious [Huq et al. 2013; Huq 2013]. These approaches

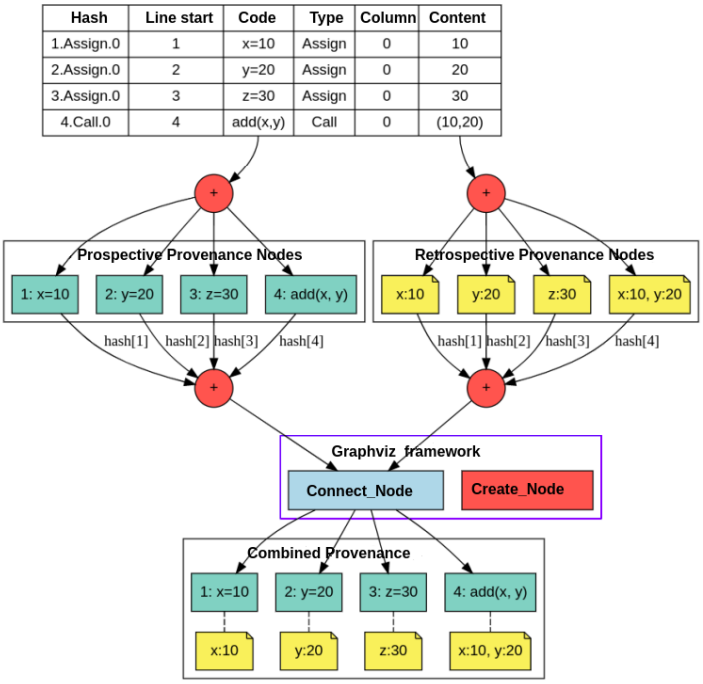


Fig. 6. Connecting diagram nodes.

were selected because they are capable of generating diagrams representing the prospective provenance of Python scripts without requiring any user intervention, such as including manual annotations. In the second phase, we compare P+RProv with the use of scripts alone. Finally, in the third phase, we compare the P+RProv diagrams using only the prospective provenance visualization versus the combined provenance. In this sense, the third phase addresses the need to compare the usefulness of using combined provenance diagrams to understand details about the script results after its execution.

4.1 Materials and Methods

4.1.1 *Research Questions.* The objective of this experimental study is to answer the following **research questions** regarding the effectiveness and efficiency of P+RProv.

—**RQ1 (effectiveness)**: Do P+RProv diagrams allow users to be more effective at analyzing and understanding the structure of a script when compared to other similar approaches?

—**RQ2 (effectiveness)**: Do P+RProv diagrams allow users to be more effective at analyzing and understanding the structure of a script when compared to not using this type of tool (only using the script code)?

—**RQ3 (effectiveness)**: Do the P+RProv combined provenance diagrams allow users to be more effective in analyzing and understanding the script when compared to using only the prospective provenance diagrams?

—**RQ4 (efficiency)**: Do P+RProv diagrams allow users to be more efficient at analyzing and understanding the structure of a script when compared to other similar approaches?

—**RQ5 (efficiency)**: Do P+RProv diagrams allow users to be more efficient at analyzing and understanding the structure of a script when compared to not using this type of tool (only using the script code)?

—**RQ6 (efficiency)**: Do the P+RProv combined provenance diagrams allow users to be more efficient in analyzing and understanding the script when compared to using only the prospective provenance diagrams?

4.1.2 *Experiment Design.* To answer these research questions, the online experiment was divided into three phases. In **Phase 1**, we compared the efficiency and effectiveness of P+RProv with the diagrams generated by Visustin and ProvenanceCurious, aiming at answering RQ1 and RQ4. To do this, we created three tasks ($A1$, $A2$, and $A3$) that use code diagrams instead of source code. These diagrams were based on the examples of prospective provenance diagrams described in [Huq 2013]. Using those as a basis, we generate code diagrams for each of the three evaluated approaches using the Python scripts that correspond to the diagrams presented in [Huq 2013]. In **Phase 2**, we created two tasks ($B1$ and $B2$). In the first task, the participants received a Python script and needed to answer a single question about it, which was related to the return value of one of the script's functions. The second task used the diagram produced by P+RProv. The participants needed to analyze the code diagram and answer a specific question about the diagram (identify a variable value in a diagram session). To mitigate learning bias in this case, we did not use the same code for both tasks. Instead, the script and the diagram were very similar, but not equal, with small changes in some functions and variable values. In **Phase 3**, we compared the efficiency and effectiveness of P+RProv when we use prospective provenance only and when we use combined provenance diagrams. For this phase, we created six tasks ($C1$, $C2$, $C3$, $C4$, $C5$, and $C6$) that use diagrams with tags (representing combined provenance) and unmarked diagrams (only prospective provenance).

The experiment was conducted online (due to the COVID-19 pandemic), using forms and images. To prevent participants from copying the codes into a Python interpreter, no participant received codes or diagrams in textual form. All script codes and diagrams were provided as images only. The participants answered the questions of each task using text fields on the forms. To prevent learning bias and problems with time measurements, the participants were prevented from returning to previous tasks once they finished a given task.

We used the Latin square [Juristo and Moreno 2001] design for the three treatments (Provenance-Curious, P+RProv, and Visustin) of Phase 1, the two treatments (script code and P+RProv diagram) of Phase 2, and the two treatments (prospective provenance and combined provenance diagram) of Phase 3, as shown in Tables I, II, and III. Thus, for Phase 1, we divided the participants into three groups (X_1, X_2, and X_3), in Phase 2, the participants were divided into two groups (Y_1 and Y_2) so that $\{X_1 \cup X_2 \cup X_3\} = \{Y_1 \cup Y_2\}$, and in Phase 3, the participants were divided into two groups (Z_1 and Z_2). To produce these groups, we used the responses of a characterization questionnaire and a heuristic for distributing participants, trying to obtain homogeneous groups in all phases of the experiment. This heuristic uses the round-robin algorithm and prioritizes experiments with missing participants with certain characterization profiles to prevent groups from becoming too heterogeneous or unbalanced. Also, following the Latin square design, each task for a given group used a different tool. For instance, group X_1 used P+RProv to perform task $A1$, Provenance Curious to perform task $A2$, and Visustin to perform task $A3$. For the other groups, the configuration changed in accordance with the Latin square design. A similar procedure was done for Phase 2 and Phase 3. The group names reflect the Latin square design, with X_i referring to the groups of Phase 1, Y_i referring to the groups of Phase 2, and Z_i referring to the groups of Phase 3 of the experiment. Since Phase 3 was conducted months later than Phases 1 and 2, the participants of Phase 3 are different from those of Phases 1 and 2.

Additionally, each task had a different difficulty level. Table IV and Table V show the subjects covered in the tasks of this experimental study. Each task focuses on basic knowledge of the Python syntax, such as conditional structures, loops, functions, function calls, variables, and parameters. Table VI shows the types of diagrams used in each phase of the study. In Phase 1, we use only prospective provenance diagrams to compare P+RProv with other tools. In Phase 2, we use combined

Table I. Latin square design for Phase 1

Groups	**Task** $A1$	**Task** $A2$	**Task** $A3$
X_1	Pros.Prov	Prov.Curious	Visustin
X_2	Visustin	Pros.Prov	Prov.Curious
X_3	Prov.Curious	Visustin	Pros.Prov

Table II. Latin square design for Phase 2

Groups	**Task** $B1$	**Task** $B2$
Y_1	Code	Diagram
Y_2	Diagram	Code

Table III. Latin square design for Phase 3

Groups	**Task** $C1$	**Task** $C2$	**Task** $C3$	**Task** $C4$	**Task** $C5$	**Task** $C6$
Z_1	P	P	P	P+R	P+R	P+R
Z_2	P+R	P+R	P+R	P	P	P

provenance diagrams to compare with the pure Python scripts. In Phase 3, we used prospective provenance diagrams and compared them to the combined provenance diagrams.

Figure 7 shows how the experiment was designed and how the participants interact with it. For the tasks of Phase 1, the participant receives an image of a diagram generated by one of the tools in this study (Step 1 of tasks of Phase 1 in the Figure). When opening a task, a (hidden) timer is automatically started. This timer runs until the participant finishes the task. After looking at the diagram, the participant must write the Python code that corresponds to that diagram in a text field on the response form (Step 2 of tasks of Phase 1 in the Figure). In Phase 2, the participants receive an image of a diagram generated by P+RProv or of a source code in Python (Step 1 of tasks of Phase 2 in the Figure). Then, the participants must look at the image and answer the corresponding questions (Step 2 of tasks of Phase 2 in the Figure). For Phase 3, participants receive an image generated by P+RProv based on either prospective provenance diagrams or combined provenance diagrams (Step 1 of the Phase 3 tasks in the Figure). Thereby, the participants must look at the diagram and answer the corresponding questions (Step 2 of the Phase 3 tasks in the Figure). It is worth mentioning that the timer is individual for each task. APPENDIX A shows details of each of the phases of the experiment, including the initial phase of answering a characterization form, giving consent, etc., and a final phase of answering a questionnaire to provide impressions of the three phases of the experiment.

The tasks of each phase are as follows.

Phase 1:

—$A1$: Write the Python code corresponding to Diagram D_1;
—$A2$: Write the Python code corresponding to Diagram D_2;
—$A3$: Write the Python code corresponding to Diagram D_3;

Phase 2:

—$B1$: Carefully observe the diagram below. Write the result produced by the red node of the graph, based on the following input: $X =$ "ab", $Y =$ "Aa";
—$B2$: Pay close attention to the following Python script. Write the result produced by line 33 based on the following input: $idA = [1, 0, 1]$, $idB = [2, 0, 3]$;

Table IV. Python knowledge required in each task of the experiment (Phase 1 and Phase 2)

Phase 1			**Phase 2**	
Task $A1$	**Task** $A2$	**Task** $A3$	**Task** $B1$	**Task** $B2$
if-else	loops	functions,calls	all	all

Table V. Python knowledge required in each task of the experiment (Phase 3)

Phase 3					
Task $C1$	**Task** $C2$	**Task** $C3$	**Task** $C4$	**Task** $C5$	**Task** $C6$
if-else	loops	functions,calls	all	for,if-else	all

Table VI. Types of provenance diagrams used in each Phase.

Diagram type	**Phase 1**	**Phase 2**	**Phase 3**
Prospective Provenance	Yes	No	Yes
Combined Provenance	No	Yes	Yes

Phase 3:

—$C1$: Observe the diagram and indicate the content of variable *VOut1* when *VGiven = 10*, *value1 = 10*, *value2 = 30* and *value3 = 90*;

—$C2$: Observe the diagram and indicate the value of the *SUM* variable at each execution of the loop;

—$C3$: Observe the diagram and write what is the value returned by variable Z at the indicated point;

—$C4$: Observe the diagram and write what is the value of variable Z (after executing the mutation function) at the indicated point;

—$C5$: Observe the diagram and write what is the value of the variable *COUNT* at the indicated point;

—$C6$: Observe the diagram and write what is the value of the variable *DATA* at the indicated point;

To evaluate the participants' answers to these tasks, we calculate the **effectiveness** using the number of correct responses obtained for each task. To calculate the effectiveness, for each correct answer, we add one point (1 point). Incorrect answers get zero points (0 points). In contrast, we measure **efficiency** as the total time it took for a participant to correctly answer a task (duration). Answers of participants that got at least one incorrect response in the same phase of the experiment were discarded. The rationale for discarding data from participants that produced incorrect answers is that one can be fast to provide wrong answers, and so this would incorrectly affect the efficiency measurements.

As tasks $B1$ and $B2$ of Phase 2 are the same (a task comparing the script code and diagram, just applied in different orders), we combine the obtained results. This way, in our analysis, task B corresponds to the union of the results of $B1$ and $B2$. This union does not interfere with the results, it only facilitates the statistical analysis of this study.

4.1.3 *Participants.* To answer our research questions, we invited participants by sending direct emails to some people, and also sending general emails to student groups from the various graduate and undergraduate courses offered by the Department of Computer Science at Universidade Federal Fluminense. We also invited external people (from other institutions and freelance developers) with some knowledge of Python. For Phase 1 and Phase 2, 42 people accepted to participate in this study, of which 20 are undergraduate students, 8 have a bachelor's degree, 7 are master's students, 3 are Ph.D. candidates, and 4 are Ph.D. Besides, 5 of them are from other knowledge areas, such as Economy Sciences, Manufacturing Engineering, Linguistics, and Biomedical Engineering. Figure 8 shows the

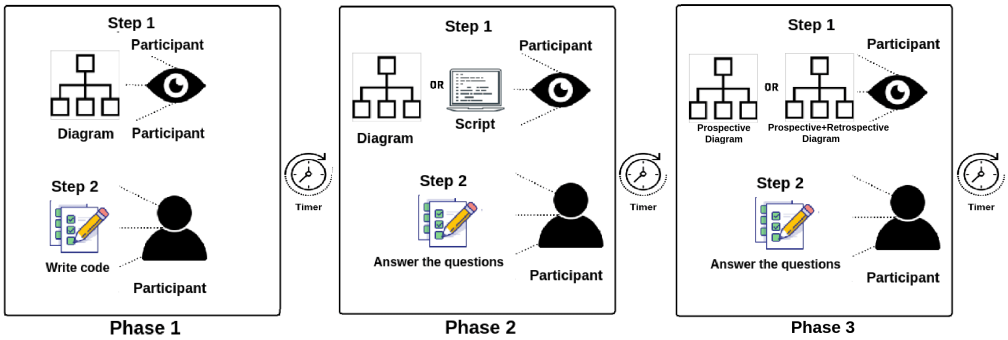


Fig. 7. Experiment Design.

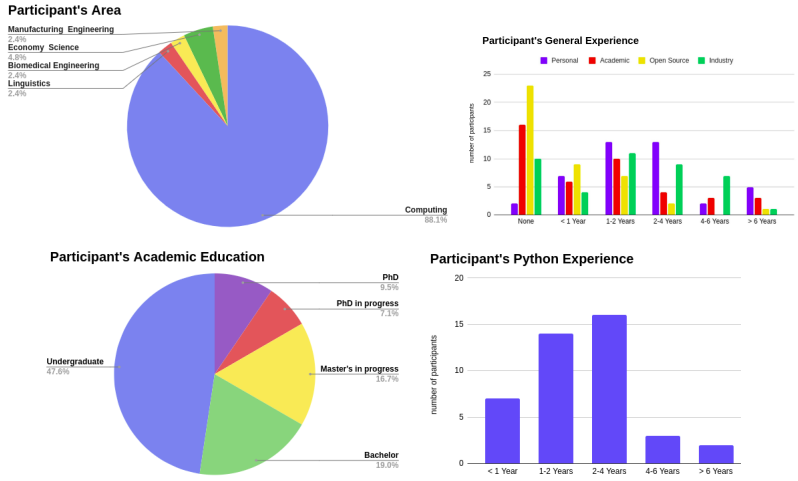


Fig. 8. Participants characterization (Phase 1 and Phase 2).

distribution of participants based on the characterization questionnaire of the experiment in Phase 1 and Phase 2. We conducted Phase 3 separately, and 34 people accepted to participate, of which 15 are undergraduate students, 1 have a bachelor's degree, 5 are master's students, 2 have a master's degree, 8 are Ph.D. candidates, and 3 are Ph.D. Besides, 4 of them are from other knowledge areas, such as Chemical Engineering, Biotechnology, Communication, and Business Administration. Figure 9 shows the distribution of participants of Phase 3.

When conducting the experimental study, we did not inform participants of the criteria for dividing the groups or group identification. To avoid any bias, we did not disclose which approach the participants were evaluating in each task. Also, we did not inform the criteria we used for evaluating the answers to the questions. When starting the study, participants received general information such as the number of tasks, the experimental study description, and instructions for submitting response forms. Before the definitive study, we applied four pilot experiments (in different sessions) with students at Universidade Federal Fluminense. We use these pilot experiments to adjust the questions for clarity and to identify problems during their execution (points of failure). As a result of these pilot experiments, we made four modifications in the experiment design to minimize the impacts of threats to validity (which are described in Section 4.3).

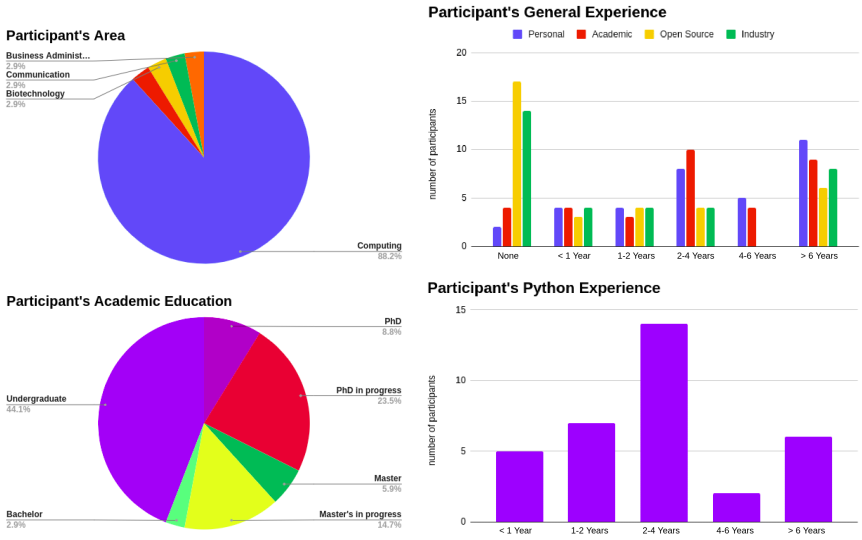


Fig. 9. Participants characterization (Phase 3).

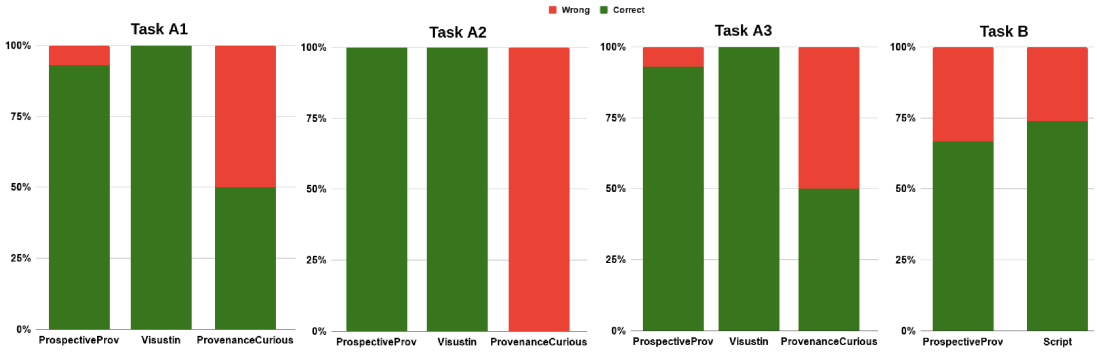


Fig. 10. Number of Correct Answers per Task (Phase 1 and Phase 2).

4.2 Results and Discussion

4.2.1 *Effectiveness.* We analyze effectiveness based on the total number of correct answers to each task. Figure 10 shows the bar graphs of the total number of correct answers of each task in this experimental study. For tasks of Phase 1 ($A1$, $A2$, and $A3$), we considered a correct answer only if the participant correctly replicated the source code. This way, we evaluate the answers based on the code correctness, analyzing whether the Python code in the participant's answers correctly represents the diagram presented in the task. Figure 10 shows that Visustin obtained the highest total of correct answers when compared to the other two approaches. In the graph, the values are defined between zero and 1.0, where 1.0 indicates 100% of correct answers. ProvenanceCurious did not score on task $A2$, as all participants presented wrong answers to this task when using ProvenanceCurious. Also, ProvenanceCurious had a higher error rate in all tasks when compared to the other approaches.

Similarly, for the tasks of Phase 2, we considered an answer as correct when the participant correctly identified the values returned by the functions using the P+RProv diagram or Python code. In task B ($B_1 \cup B_2$), the purpose is to compare the efficiency of using code diagrams when compared to using the source code to represent scripts. Figure 10 shows that participants scored higher when they

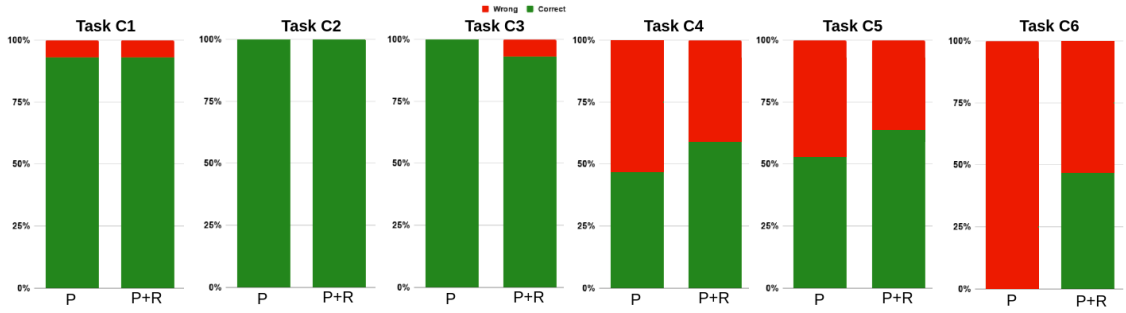


Fig. 11. Number of Correct Answers per Task (Phase 3).

Table VII. *P-value* for the Fisher test for the total number of correct answers. The null hypothesis is that there is no difference between the compared approaches. Blue values indicate *p-value* $<= \alpha$, with $\alpha = 0.05$

2x2 Comparison	Phase	Task	*p-value*
P+RProv and Visustin	Phase 1	A1	≈ 1.0000
		A2	≈ 1.0000
		A3	≈ 1.0000
P+RProv and ProvenanceCurious	Phase 1	A1	0.0329
		A2	<0.0001
		A3	0.0329
Visustin and ProvenanceCurious	Phase 1	A1	0.0058
		A2	<0.0001
		A3	0.0058
Diagram and Script	Phase 2	B	0.6337
Prospective and Combined Provenance	Phase 3	C1	≈ 1.0000
		C2	≈ 1.0000
		C3	≈ 1.0000
		C4	0.7319
		C5	0.7283
		C6	0.0027

used the source code. It is important to note that the difference between the total number of correct answers of tasks performed using P+RProv diagrams and the script is only 3 points (and we had 42 participants). In addition, 60% of participants from non-computational areas felt more comfortable with the P+RProv diagrams than with the scripts, according to the answers they provided in the feedback questionnaires.

In Phase 3, the objective is to compare the effectiveness of prospective provenance diagrams and combined provenance diagrams in the analysis of Python scripts. Figure 11 shows the number of correct answers per task. The qualitative analysis of Phase 3 follows the same criteria we used for Phase 1 and Phase 2. In this analysis, we observed that 67.7% of participants in the experiment group prefer to use the combined provenance diagrams.

To measure the statistical significance of the total number of correct answers, we applied the Fisher test to compare the approaches 2x2 [Finney 1948]. Table VII shows the Fisher test results. In this case, there is a statistically significant difference when we compare P+RProv and ProvenanceCurious in tasks $A1$, $A2$, and $A3$, a behavior that also occurs between Visustin and ProvenanceCurious. The difference in the results of P+RProv and Visustin is not statistically significant. Thus, we conclude that both are equally effective. The same occurs for the results of P+RProv and scripts in Phase 2. In the case of the prospective provenance diagrams and the combined provenance diagrams (Phase 3), the *p-value* shows that the difference in the results is not statistically significant for the observed

tasks, except for task $C6$.

RQ1: Do P+RProv diagrams allow users to be more effective at analyzing and understanding the structure of a script when compared to other similar approaches?

Answer: *Participants who performed the tasks using P+RProv obtained the second-highest number of correct answers, except for task A2, where Visustin and P+RProv have similar success rates. However, there is no statistically significant difference between the results of Visustin and P+RProv, and they were both more effective than ProvenanceCurious (Table VII). Visustin and P+RProv both use diagrams based on flowcharts, while ProvenanceCurious uses provenance notations. Thus, participants performed worse using provenance notations. P+RProv has the advantage of using execution provenance data to build the diagram, and so it guarantees that the diagram reflects a correct code, while Visustin is not capable of guaranteeing that. Also, unlike Visustin, P+RProv is able to include retrospective data in the diagram.*

RQ2: Do P+RProv diagrams allow users to be more effective at analyzing and understanding the structure of a script when compared to not using this type of tool (only using the code)?

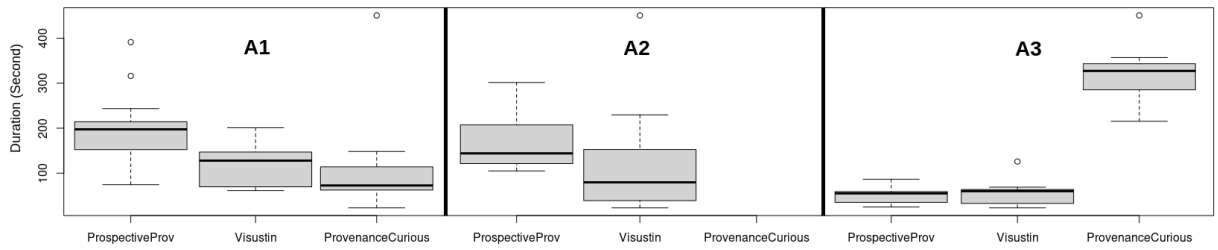
Answer: *Participants who performed the tasks using only the code obtained higher scores than participants who used the code diagrams. However, this difference is not statistically significant (Table VII). We thus conclude that both approaches are equally effective. It is worth noting that, based on the final feedback questionnaire, 60% of the participants from areas different from Computer Science expressed a greater preference for diagrams instead of script code.*

RQ3: Do the P+RProv combined provenance diagrams (retrospective + prospective) allow users to be more effective in analyzing and understanding the script than only prospective provenance?

Answer: *In more complex tasks (as tasks C4, C5, and C6), participants who performed the tasks using the combined provenance diagrams obtained more correct answers. However, that difference is not statistically significant, with the exception of task C6, where* p-value $<= \alpha$. *It is important to emphasize that none of the participants correctly answered the C6 task when they used only the prospective provenance diagram. That indicates combined provenance can help understand more complex problems, where manually parsing the script becomes infeasible.*

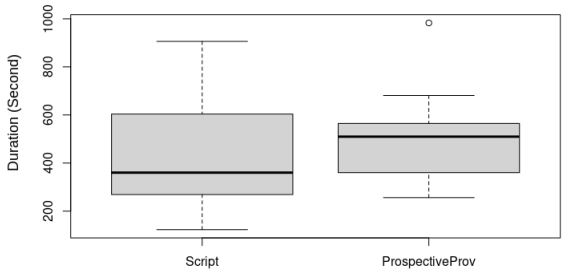
4.2.2 *Efficiency.* To measure efficiency, we also need to take effectiveness into account, since there is no point in finishing the tasks fast, but providing wrong answers. Thus, to analyze efficiency, we discard the wrong answers and compare the total time participants took to finish each task that was answered correctly – we call this variable *duration*. This means that ProvenanceCurious will be highly affected, with several data points discarded (including all of them for task $A2$), but this is not a problem since it is not as effective as P+RProv and Visustin (Figure 10). Figure 12 shows the boxplots for the duration variable and summarizes the obtained results for the tasks of Phase 1, while Figure 13 shows the results for the task of Phase 2, and Figure 14 shows the results for the tasks of Phase 3.

To test if there is a statistically significant difference in the duration of each task, we use the Mann-Whitney test to compare the approaches 2x2. This test compares the difference of the distribution in a continuous variable (duration) for two populations. Table VIII shows the results. For the tasks of Phase 1, note that there is a significant difference between ProvenanceCurious and the other approaches in most of the tasks, thus leading to the conclusion that it is less efficient than P+RProv and Visustin. When comparing P+RProv with Visustin, there is a significant difference only in Task $A1$, where participants that used Visustin performed better than those who used P+RProv. In the remaining two tasks, there is no significant difference.



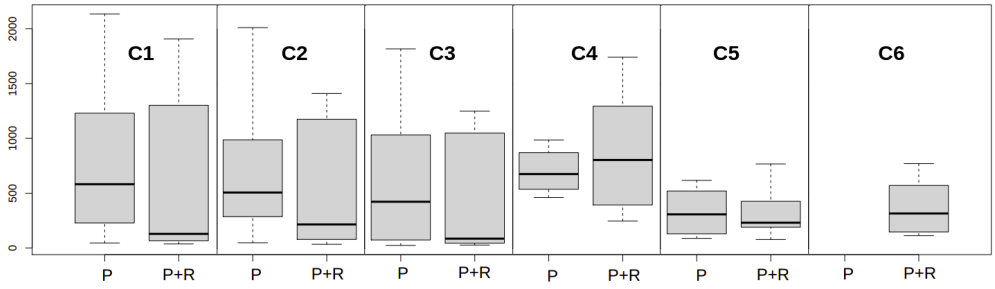
	Task $A1$			Task $A2$			Task $A3$		
Metrics	**PProv**	**Visustin**	**PCurious**	**PProv**	**Visustin**	**PCurious**	**PProv**	**Visustin**	**PCurious**
average	204.50	117.69	128.40	104.49	75.56	—	114.69	119.36	617.92
std	80.79	47.79	146.94	39.54	68.91	—	32.47	48.88	139.30

Fig. 12. Analysis of the Duration of tasks of Phase 1, considering only correct answers.



	Task B	
Metrics	**Scripts**	**P+RProv**
average	428.7097	483.9000
std	206.0263	160.7115

Fig. 13. Analysis of the Duration of tasks of the task of Phase 2, considering only correct answers.



	Task $C1$		Task $C2$		Task $C3$		Task $C4$		Task $C5$		Task $C6$	
Metrics	**P**	**P+R**	**P**	**P+R**	**P**	**P+R**	**P**	**P+R**	**P**	**P+R**	**P**	**P+R**
average	743.37	519.88	726.65	510.42	625.77	466.57	701.12	845.80	335.34	324.09	—	368.50
std	636.84	597.89	631.35	557.50	608.41	527.16	196.77	494.67	208.15	211.89	—	256.42

Fig. 14. Analysis of the Duration of tasks of Phase 3, considering only correct answers.

In the comparison between P+RProv and scripts (Phase 2 of the experiment), although there is a small difference in the average time for participants who used the script code, there is no statistically significant difference. We emphasize that even with a larger proportion of participants from the Computer Science area (who are supposedly more familiar with scripts), the difference in the results is not statistically significant. Besides, in Phase 3 of the experiment, when comparing Prospective Provenance and Combined Provenance, there is no significant difference between the approaches – participants had a similar performance.

Table VIII. *P-value* for the Mann-Whitney test for the duration variable. The null hypothesis is that there is no difference between the compared approaches. Blue values indicate *p-value* $<= \alpha$, with $\alpha = 0.05$

2x2 Comparison	Phase	Task	*p-value*
P+RProv and Visustin	Phase 1	A1	0.0024
		A2	0.0601
		A3	0.7871
P+RProv and ProvenanceCurious	Phase 1	A1	0.0214
		A2	—
		A3	0.0003
Visustin and ProvenanceCurious	Phase 1	A1	0.4354
		A2	—
		A3	0.0003
Diagram and Script	Phase 2	B	0.1362
Prospective and Combined Provenance	Phase 3	C1	0.3571
		C2	0.1222
		C3	0.2604
		C4	0.7897
		C5	0.8820
		C6	—

RQ4: Do P+RProv diagrams allow users to be more efficient at analyzing and understanding the structure of a script when compared to other similar approaches?

Answer: *It depends on the task. Participants who used Visustin performed better than P+RProv on task A1 in terms of efficiency. However, for the remaining tasks, P+RProv is as efficient as Visustin. It is worth noting that task A1 is the simplest of all. For more complex tasks (A2 and A3), there is no significant difference between the two approaches. When compared to ProvenanceCurious, P+RProv is more efficient. In the particular case of the task A2, ProvenanceCurious is not efficient since there were no correct answers (all samples for that task were discarded).*

RQ5: Do P+RProv diagrams allow users to be more efficient at analyzing and understanding the structure of a script when compared to not using this type of tool (only using the code)?

Answer: *No, our analysis concluded that there is no significant difference between the results. However, according to the final questionnaire, participants from other areas of knowledge (excluding Computer Science) felt more comfortable answering the questions using diagrams.*

RQ6: Do the P+RProv combined provenance diagrams allow users to be more efficient in analyzing and understanding the script when compared to using only prospective provenance diagrams?

Answer: *No, our analysis concluded that there is no significant difference for the duration of tasks of participants who used prospective and combined provenance diagrams.*

4.3 Threats to Validity

During the planning of this experimental evaluation, we sought to avoid threats that could impact or limit the validity of our results. In this section, we describe the threats we identified in this experimental evaluation.

Due to the COVID-19 pandemic, our study occurred using online forms. Participants could respond to this study at any time, depending on their availability. Therefore, we cannot guarantee that these participants completed that study in a silent environment and without interruptions during the process. However, we use statistical methods to analyze and remove outliers in order to minimize this threat.

We provide participants with different tasks to minimize the learning effects at each step of this study. Besides, we did not inform which of the approaches would be evaluated to avoid any bias in the candidates' responses. Even using these treatments, we cannot guarantee the complete elimination of these effects and threats.

Another threat is related to the characterization of the participants. We developed a heuristic using a round-robin algorithm to divide the participants into homogeneous groups. However, we cannot guarantee that everyone answered the characterization questionnaire correctly. This way, incorrect or unfilled answers in the characterization questionnaire can lead to an unbalance of the study groups. Most participants are undergraduate students, which limits the representativeness of the population in this study. In order to minimize these effects, we previously analyzed the data of each participant, then decided to eliminate any candidates with inconsistent responses in the characterization questionnaire.

We have 42 volunteer participants (in Phase 1 and Phase 2) and 34 volunteer participants in Phase 3. Although this is a small number, 30 participants are considered a sufficiently large sample for a controlled experiment with people [Juristo and Moreno 2001]. Thus, this number of samples increases the statistical validity of the conclusions we obtained. Another threat is the selection of participants of phase 3, as some of them may have already participated in the previous phases, as we can not verify the participant's identification.

Finally, as this study is online (using web forms), we cannot monitor the activities of the participants until they submit the response to each task. Thus, the participants might obtain advantages in performing the tasks using external consultations, which is a significant threat to this study. To try to minimize this threat, each task had a time limit for completion (which was not reached by any of the participants). Also, the source codes were provided as images to try to prevent the participants from copying codes and running them on a local machine to answer the questions of our study.

5. CONCLUSION

This paper presents P+RProv, an approach to help scientists to understand the code structure using prospective and prospective+retrospective provenance diagrams. We use flowchart diagrams instead of provenance notations as the principal way to represent the structure of Python scripts. Our approach has several advantages when compared to existing related work: (i) it guarantees the diagram represents a correct script, since it is generated from execution provenance data that was captured by noWorkflow [Pimentel et al. 2017]; (ii) it is able to include retrospective data on the diagram, which none of the existing approaches is able to do.

We conducted an experimental evaluation to answer four research questions related to the effectiveness and efficiency of P+RProv. The main goal of the evaluation is to assess whether our approach is effective and efficient in helping people to understand and analyze the structure of Python scripts. When comparing P+RProv with related approaches, we conclude that P+RProv is more effective and efficient than ProvenanceCurious. As for Visustin, there were no significant differences in terms of effectiveness. For efficiency, Visustin is more efficient than P+RProv on task $A1$, which is the simplest of all tasks. For more complex tasks, there is no statistically significant difference in terms of efficiency. Additionally, Visustin does not guarantee the correctness of the Python code that was used to generate the diagram (Visustin diagrams are generated based only on the syntax of the script). There is no prior verification or analysis of the Python script to guarantee it is correct and that it produces the expected results. Visustin and P+RProv use diagrams based on algorithm flowcharts, which may explain the shorter time participants took to complete the tasks in our experimental evaluation when compared to ProvenanceCurious. In fact, flowchart diagrams are more common than provenance notation, and the chances of participants having seen similar diagrams are high.

ProvenanceCurious obtained a result well below that of other approaches, especially in tasks with more elaborate diagrams and with more edges. According to the participants, the tasks completed

using ProvenanceCurious were more time-consuming, less intuitive, and with fewer correct answers. Most participants found the ProvenanceCurious diagrams hard to understand when compared to Visustin and P+RProv.

When comparing scripts and P+RProv, there is no statistical difference both in terms of effectiveness and in terms of efficiency. However, according to the final questionnaire, participants from other areas of knowledge (excluding Computer Science) felt more comfortable answering the questions using P+RProv. Since the majority of participants are from the Computer Science area, additional evaluation with a more diverse population is essential.

As for the combined provenance diagrams generated by P+RProv, there was a statistically significant result for the more complex and larger script (task $C6$). Furthermore, there were also positive results in the qualitative research, where 67.7% of the participants felt more comfortable using the combined provenance diagrams.

As future work, we plan to support other programming languages, /newimprove P+RProv's visualization methods, and improve the methods that combine prospective and retrospective provenance in order to enhance the users' understanding of scripts. We also intend to replicate our experimental evaluations with the inclusion of new tools that support different programming languages, a comparison of Visutin and scripts, and the inclusion of harder tasks that were unfeasible in a remote experiment setting.

REFERENCES

Davison, A. P. Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. *Computing in Science & Engineering* 14 (4): 48–56, 2012.

Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., and Woodhull, G. Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools. In *Graph Drawing Software*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 127–148, 2004.

Finney, D. The Fisher-Yates test of significance in 2X2 contingency tables. *Biometrika* 35 (1/2): 145–156, 1948.

Freire, J., Koop, D., Santos, E., and Silva, C. T. Provenance for computational tasks: A survey. *Computing in Science & Engineering* 10 (3): 11–21, 2008.

Herschel, M., Diestelkamper, R., and Lahmar, H. B. A survey on provenance: What for? what form? what from? *VLDB Journal* 26 (6): 881–906, 2017.

Huq, M. R. *An inference-based framework for managing data provenance*. Ph.D. thesis, University of Twente, Enschede, Netherlands, 2013.

Huq, M. R., Apers, P. M. G., and Wombacher, A. ProvenanceCurious: a tool to infer data provenance from scripts. In *International Conference on Extending Database Technology (EDBT)*. ACM, Genoa, Italy, pp. 765–768, 2013.

Juristo, N. and Moreno, A. M. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Spain, 2001.

Lerner, B. and Boose, E. R. RDataTracker: collecting provenance in an interactive scripting environment. In *Workshop on the Theory and Practice of Provenance (TaPP)*. USENIX Association, Cologne, Germany, pp. 1–4, 2014.

Linhares, H., Pimentel, J. a. F., Kohwalter, T., and Murta, L. G. P. Provenance-Enhanced Algorithmic Debugging. In *Brazilian Symposium on Software Engineering (SBES)*. ACM, Salvador, Brazil, pp. 203–212, 2019.

McPhillips, T., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, K., Cao, Y., Chirigati, F., Dey, S., Freire, J., et al. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. *International Journal of Digital Curation* 10 (1): 298–313, 2015.

Murta, L., Braganholo, V., Chirigati, F., Koop, D., and Freire, J. noWorkflow: Capturing and Analyzing Provenance of Scripts. In *Provenance and Annotation Workshop (IPAW)*. Springer, Cologne, Germany, pp. 71–83, 2014.

Nassi, I. R. and Shneiderman, B. Flowchart techniques for structured programming. *ACM SIGPLAN Notices* 8 (8): 12–26, 1973.

Oy, C. A. Visustin flowchart generator. https://www.aivosto.com/visustin.html, 1997. Accessed: 2021-05-20.

Ozgur, C., Colliau, T., Rogers, G., Hughes, Z., and Bennie, E. Matlab vs. python vs. r. *Journal of data science: JDS* vol. 15, pp. 355–372, 07, 2017.

Pimentel, J. a. F. *Provenance from Scripts*. Ph.D. thesis, Universidade Federal Fluminense, Niterói, RJ, 2021.

Pimentel, J. F., Freire, J., Murta, L., and Braganholo, V. A Survey on Collecting, Managing, and Analyzing Provenance from Scripts. *ACM Computing Surveys (CSUR)* 52 (3): 1–38, 2019.

Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. noWorkflow: a Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts. *Proceedings of the VLDB Endowment* vol. 10, pp. 1841–1844, 2017.

Simmhan, Y. L., Plale, B., and Gannon, D. A survey of data provenance in e-science. *ACM SIGMOD Record* 34 (3): 31–36, 2005.

Weintraub, P. G. The importance of publishing negative results. *Journal of Insect Science* 16 (1): 1–2, 2016.

APPENDIX A. DETAILS OF THE EXPERIMENTS

This section describes the flow of our online experiment in details. Details about diagrams and codes used in the experiment are available at `https://github.com/dew-uff/prospective-prov/`

Initial Phase

(1) The participant answers a characterization form.
(2) The participant is placed on a group based on her academic experience and Python experience.
(3) The participant receives on-screen study instructions and the consent form.
(4) The participant receives basic instructions to perform the tasks.

Phase 1

(1) The participant receives instructions for the tasks of Phase 1.
(2) The participant starts the task $A1$ - she receives an image of a diagram and the question that must be answered. The hidden timer is started. She provides the answer to the task and clicks a button to go to the next task, when the timer is finished. After this point, it is not possible to come back to this task anymore.
(3) Step 2 above repeats for tasks $A2$ and $A3$.
(4) After finishing task $A3$, the participant receives an evaluation questionnaire. The participant evaluates the questions she has just answered.

Phase 2

(1) The participant receives instructions for the tasks of Phase 2.
(2) The participant starts the task $B1$ - she receives a diagram or script and the question that should be answered. The hidden timer is started. She provides the answer to the task and clicks a button to go to the next task, when the timer is finished. After this point, it is not possible to come back to this task anymore.
(3) Step 2 above repeats for task $B2$.
(4) The participant receives an evaluation questionnaire for tasks of Phase 2.
(5) The participant submits her answers, and the experiment is finished.

Phase 3

(1) The participant receives instructions for the tasks of Phase 3.
(2) The participant starts the task $C1$ - she receives a diagram with prospective or combined provenance data. The hidden timer starts. She provides the answer to the task and clicks a button to go to the next task when the timer ends. After that point, it is no longer possible to return to that task.

(3) Step 2 above repeats for task $C2$, $C3$, $C4$, $C5$ and $C6$.
(4) The participant receives an evaluation questionnaire for tasks of Phase 3.
(5) The participant submits her answers, and the experiment is finished.