Why-Diff: Exploiting Execution Provenance to Answer the Difference Between Two Computational Results

Priyaa Thavasimani

***Abstract*—Provenance plays an important role in computational reproducibility. Nowadays, almost all the workflow management systems (WfMS) are collecting both the prospective and retrospective provenance which enables researchers to reproduce the experiments at ease. Researchers either use 'known' scientific artefacts in their experiments to achieve expected experimental output or curious to know the 'unknown' scientific artefacts which caused a different result. In this paper, we introduce a system Why-Diff which takes advantage of the retrospective provenance captured by WfMS to tell the difference between two computational results and also to help researchers to achieve expected impact out of their experiments.**

I. Introduction

Researchers conduct experiments to test a hypothesis by executing scripts, stand-alone programs or workflows. They often reproduce experiments to achieve their expected results and the artefacts of the successfully reproduced computations play major role in the extension of the research. Despite successful reproducibility, many of the science experiments outputs unexpected results. Dealing with large datasets is a complex task and exploiting the available datasets to answer a scientific question is important in any kind of Research. Not always, a Researcher get the expected results and this is because of using different edition and/or version of the software packages. In our previous work [1], we considered single cause for the difference in the results of original and reproduced computations. In real world, a workflow is reproduced with multiple variations like input files has been changed, software/library has been updated, external data sources has been modified, new features has been added to the workflow, and old features has been removed from the workflow. And it is possible that all these changes can happen at the same time thus multiple variations could affect the final result. To achieve desired result of the reproduced computations, it is essential to keep track of the artefacts used throughout the workflow.To achieve this, the researcher need to document the steps involved in the computation so that others could retrace the steps documented and reproduce similar results. These steps of documenting and reproducing are made easy once the application is represented as workflow. There are existing workflow management systems like [2], Pegasus [3], Galaxy [4], VisTrails [5], Kepler [6], Taverna [7], which automates the computation by mapping the abstract workflows into the underlying resources. Successful reproducible science requires the ability to compare results across variations of an e-science experiment, and to provide explanations for sets of results that differ across executions. We have used the notations W, X, D and Y to refer Workflow (set of Workflow blocks [w_1, w_2, ...]), Input (set of inputs [x_1, x_2,...]), Dependency (set of dependencies [d_1, d_2, ...) and Output (set of Outputs [y_1, y_2, ...]) respectively. The cause for the difference could be only W but not D,X, or only D but not W,X or X but not W,D or combinations of the these elementary variations, for instance change both W and D, W and X, etc. Having this as underlying framework, we aim to verify and validate the results of successfully reproduced computational results by investigating the computational artefacts and execution provenance.

II. Background

In previous studies [8]–[15], the terms "Reproducibility" and "Replication" are widely discussed, of which most researchers emphasise the importance of reproducibility.

Researchers classify provenance captured by the system into two basic types [16] : Prospective and Retrospective. Prospective provenance captures the abstract specification of the computational tasks like library, data or any form of dependencies which make the Workflow to run whereas retrospective provenance captures more detailed information like execution environment, inputs, outputs and intermediate results cached.

Workflow Management System (WfMS) is used to execute "in-silico" experiments. It provides a collaborative environment for researchers to create, execute and share workflows. There are various types of Workflow Management Systems, some of which includes E-Science Central [2], Pegasus [3], Galaxy [4], VisTrails [5], Kepler [6], Taverna [7]. WfMS captures fine-grained provenance capturing Workflow evolutions and its associated datasets. Researchers take these provenance traces to ease reproducibility so that to avoid false leads by original experiments.

The comparison by existing tool Vistrail provenance detects files contents by hashes, allowing users to later check that provided data does indeed match that used in the original experiment [17]. The VisTrail team has also included persistence package feature that allows users to store input, output and intermediate data in a versioned history.

An issue with the VisTrail was the impact on reproducibility based on the updated database. To address this issue, a model was developed that integrated workflow and database provenance and enables reproducibility for workflows that interacts with relational databases [18]. Similar to database provenance, the researcher [19] tracked the dependencies of external web service and developed a Web Service Monitoring Framework. VFramework is a framework created to verify that a redeployed process performs according to expectations. This framework which was evaluated using Taverna, uses an ontology-based model [20] for description of processes and their dependencies. This framework was evaluated to check the verification and validation capabilities by re-executing the use cases in Linux and Windows based environments [21].

Galaxy is a web-based WfMS for data intensive biomedical research which supports provenance tracking through its "History" tool. Another provenance tracking system is Sumatra [22] which captures each execution in configuration management tool, allows users to tag and compares the collected provenance traces. Sumatra captures the difference between two workflows at the code-level i.e only the abstract provenance.

Almost all the above-mentioned WfMS keeps track of the evolution of experiments by version control. Further to monitoring tool that aid tracking changes in the external dependencies, it has become essential to analyse and monitor data transformations. Workflow is an orchestration of activities that would have used and generated files and activity could be database-related or web-service related, which acts up on data. Simply comparing files based on hash-value or code is not enough as there are numerous workflows run every single day. It is essential to link which piece of data that the particular workflow produced or consumed. To the best of our knowledge, there is no existing approach which traverse the execution traces of workflow to analyse the difference between two executions. The authors [23] discusses the tracking and analysing the provenance evolution from the scripts helps to reconstruct the experiment easing reproducibility. NoWorkflow captures execution provenance and provides facilities to compare two different executions by visualising low-level provenance details that shows divergence in function calls. However, NoWorkflow is not language-neutral and also captures very low-level details that lack clarity in answering the reason for different results. Vistrail [24] provides mechanism to users to perform paramter sweeps and to vary input data to compare results side by side. The model compares the specification of different workflow. However our "Why-Diff" compares the provenance of workflow executions.

PDIFFView [25] addresses the problem of comparing two workflow graphs so that to understand the provenance of result. The workflow structure is captured as series-parallel graph overlaid with well-nested forking and looping (SPFL). The comparison is made between two workflows each representing nodes as activities and edges represents data flow between activities. Provenance Difference Viewer [26] is demonstrated based on time-differencing algorithms for differencing workflow runs. Four types of path edit operations (Path insertion, Path deletion, Path expansion, Path contraction) are considered. The prototype system shows the inserted and deleted paths in the original workflow. However, while reproducing a workflow, there might not only be the possibility for insertion or deletion of workflow blocks but also replacement of one workflow block with other. Each and every workflow blocks are associated with the generic properties like title, block id, block description, block details, etc., and the variations in these properties could impact the final output of the workflow. In this, the structure of the workflow is emphasised, ignoring the scientific analysis involved in the workflow.

In the paper [27], authors addressed the comparing Workflows with Delta Tool. Delta compares workflows in three level of granularities: a high-level, clustered view; an intermediate-level view focusing workflow summaries; and a low-level detail view. They focus on image-editing workflows and so their criterias to compare two workflows includes the quality of the result, Photoshop commands, and efficiency of the workflow. Unlike image-editing workflows, comparison between data-intensive workflows should consider activities, external data sources, external libraries, input data, intermediate data, and output data which is not addressed in this paper.

The paper [28] highlights removing duplicates while merging database is a non-trivial task. One possible solution is to find the entity representing the data object. This try to create an alignment that connects pairs of nodes, in the two versions, that represent the same entity. Authors demonstrated 3 types of alignments which are trivial alignment, bisimulation alignment and similarity measure alignment on RDF graphs. We took advantage of the trivial alignment which is to match the input/output of first workflow with the input/output of second workflow irrespective of the structure of the workflow.

Just comparison does not explain the difference between two graph. Graph Visualisation plays key role in understanding the cause for the different results of the two workflows. The delta graph which we visualise as result of comparison bears some resemblance to the Direct Classification of node Attendance [29], an algorithm to find isomorphisms between graphs and subgraphs with the main difference that our why-diff highlights the unmatched nodes as subgraph. The paper applies DCA algorithm on two usecases satellite imagery ingest processing pipeline and the other a provenance capture of the network layers of large-scale distributed network applications. Exploratory and explanatory visualizations are discussed where exploratory visualizations helps researchers to explore large volumes of data and explanatory visualisation aids in expaining the provenance.

The "Why-Diff" is built with base, "PDIFF" algorithm which discusses and proposes algorithm diagnosing the divergence between two executions. It also highlights the number of data comparison function which helps to differentiate not only type and representation of data sets but also semantics of the data. Research Context: As mentioned in background study, this research set in the context of reproducible research. The ability for one researcher to validate, verify, reproduce, and repurpose other researchers results has always been at the core of scientific discourse. Computational workows involves series of steps in which data is generated, manipulated, analysed and transformed. As modern experimental science becomes increasingly data-driven and reliant upon big datasets and complex computations to generate new results, we have a chance of making science more reproducible, i.e., by appropriately managing the datasets, the experimental processes, and the associated metadata that describes both of them. Enabling reproducibility in science is therefore becoming the focus of new research within a vibrant research data management community.

Research Problem: My research addresses a specific problem in this setting. Suppose, in abstract, that you are given an experiment, described by a Workflow W with input X and additional dependencies D (these can be dependencies on external data resources, as well as on third party software libraries or system environment), which at a certain time t produced an output Y, our scientific result. A third party who wishes to reproduce P at a later time t would have to re-build, re-deploy, and re-execute P in a new environment. This is relatively easy to achieve using modern virtualisation technology (VM, Docker, etc.), but it is also not very interesting. A more challenging problem occurs when variations are introduced in the process and its dependencies: $\{W, X, D, Y\}$ —$\{W', X', D', Y'\}$, which lead to different outcomes Y' not equal to Y when the new version is executed. This may happen accidentally, i.e., when the external data sources are updated or not all requirements (some of which may not be explicit) have been met at the time P is deployed again, or intentionally, when the experimenter intends to explore variations of the original experiment. Intuitively, knowledge of the structure, execution trace and runtime dependencies on data and software are critical to answering these questions. These are all forms of metadata, which hopefully help explain the observed experimental outcomes. However there is no clear indication from the literature (please refer to background study) on just how much and exactly what type of metadata is needed, how it can be processed and analysed, and how much effort is required to collect it. These are the high level questions that my research will address.

III. Research Hypothesis

1. We explore the hypothesis that the provenance of experiment executions, in combination with the ability to compare datasets (inputs, intermediate results, outputs) provides a useful foundation for generating explanations as described above.
2. Why is this challenging? our hypothesis rests on two main assumptions: (1) that sufficiently detailed provenance can be associated to an e-science experiment and to its execution, and (2) that "diff" functions are available to compare the datasets involved in the experiment. Both these assumptions come with their challenges.
   a. Provenance comes at different levels of detail and completeness, depending on the nature of the experiment and most importantly of the runtime infrastructure and its ability to capture details of process structure as well as execution.
   b. Datasets used in e-science tend to be complex, they vary vastly across disciplines, and their interpretation is very domain- and application -specific. So it may be difficult and time-consuming to define suitable "diff" functions for the datasets that we want to compare.
3. Even if we assume, in the best case scenario, that our two requirements are fully satisfied, the problem of comparing two provenance traces may be challenging.

IV. Research Question

Q1: Whether the computation achieved the desired impact?
Q2: What would be the cause for unexpected results? Is it a single or multiple causes responsible for the difference?
Q3: How the input difference affects the output data?
Q4: What provenance must be collected and packed along with original experiment that enables the verification and validation of results of reproduced experiment with respective to results of original experiment?

V. Remaining Work

The algorithm "Why-Diff" is designed to forward and backward track the provenance of entire workflow for intentional and unintentional changes made to the reproduced Workflow blocks. The algorithm is extended to accomodate non-isomorphic cases. The initial experiments are demonstrated with varied dependencies, input and workflow blocks assuming the two Workflow graphs are topologically similar as well as disimilar. As of now, data nodes are compared based on file content instead of just file names. Initial experiments are conducted on test sets comprising of 10 eScience workflow invocations as well 10 non-isomorphic cases which will give 90 workflow comparisons are made (45+45). I am in finalising my paper for submission of the chapter III to IEEE eScience Conference. I intend to write a journal paper for concluding my PhD work. I am writing my thesis in parallel with working for the paper.

VI. The Completed Work

The initial investigations was done to confirm that the results of a reproduced computation may differ in accordance with original computation which is explained in Chapter I. The "Chapter I" discusses about the issues in computational reproducibility and also about existing tools' approach towards the problem. The "Chapter II" introduces our new system "Why-Diff" which is applicable for only isomorphic workflow graph structure. The execution provenance of a computation can be used to investigate why the results are different when compared to other. The researcher might want to understand the "cause" for the difference or "impact" that the computation achieved. The system "Why-Diff" is designed and implemented which is explained in Chapter II. The "Chapter III" discusses about the extension of our system "Why-Diff" so that to accomodate non-isomorphic cases.

VII. Time Plan

The project can perform its objective by executing the following scheduling time:

1: September, 2015 - April, 2016: Attending interdisciplinary training and coursework. Experiment with the provenance capturing tools to understand its provenance generation capability. Writing Literature Review.
2: May, 2016 to June 2016: Developing sample usecase that best describes the reproducible research. Experimented

the usecase with YesWorklow and NoWorkflow, capturing retrospective and prospective provenance. Demonstrate the work to supervisory team in presentation.

3: July, 2016 to August, 2016: Finalising project plan, submitting, and presenting annual report.
4: September, 2016 to March, 2017: Begin working on algorithm and tool to analyse the divergence between results obtained using different versions of a process (with evolving dependencies).
5: April, 2017 to August, 2017: Implementation of the algorithm and initial evaluation of the research using the ReComp project as a testbed. Continued Aim to submit a paper to a highly ranked conference. Present findings to supervisory group in presentation.
6: September, 2017 to May, 2018: Augmenting features to the algorithm such as to support non-isomorphic graphs, file-content comparison. Extended evaluation with respect to a more comprehensive set of case studies, continued publication. Publication of results in top quartile conferences and journals.
6: May, 2018 to August, 2018: Writing conferences and journal paper.
7: September, 2018 to September, 2019: Writing up thesis and final examination.

VIII. Thesis Table of Contents

In this section, a detailed table of contents for the thesis is presented with a description about each chapter. So far, I have three completed parts of my research work and they are well-documented as 3 papers.

The first paper [1] which is focused on reproducibility issues and existing tools' approach to the problem which has been published in the proceedings of 2016 IEEE International Conference on Big Data. In this paper, we present an approach, Why-Diff, to analyse the difference between two related computations by changing the artifacts and how the existing tools YesWorkflow and NoWorkflow record the changed artifacts.

In the second paper [30], we investigate the hypothesis that the metadata of a workflow execution can be used to explain why the experimenter observes different results (cause analysis). Similarly, Scientific metadata can be used to determine the impact of intentional variations that the experimenter may have injected into a new version of the workflow. We explore these two complementary cases using a simple algorithm "Why-Diff" for traversing two metadata traces in lock-step mode, which we illustrate through two human genomics data analysis workflows.

The third paper presents the extension of our previous system "Why-Diff" to accomodate the non-isomorphic workflows which is ready to be submitted for IEEE eScience 2018.

The above mentioned parts of the work will occupy three chapters of the thesis (3, 4 and 5 respectively) with number of experiments and results. The remaining chapters are as follows:

1: Introduction: The research problem will be detailed in this chapter with its justification and work motivations. In addition, full details about the main contritions of this research will be stated as well as a brief map for thesis structure.
2: Background and Literature Review In this chapter, number of sections for background and state of the art of related techniques, approaches and subjects have been written and aggregated related mainly to my PhD research. In addition, the related work sections in the papers mentioned above will be moved to this chapter which will compose an essential parts related the main work of the research.

The full table of contents for the thesis is as follows:

a: Chapter One: Introduction - Research Hypothesis - Research Question - Contributions - Thesis Structure
b: Chapter Two: Literature review - Reproducibility Research & Reproducibility issues. - Provenance - Workflow Management Systems: discusses about various Workflow Management Systems like Vistrails, Taverna and Sumatra and how each systems keep track of provenance and Workflow Differencing. - Existing tools comparing workflows provenance: This section explains about how the existing tools like YesWorkflow and NoWorkflow compares two computations' provenance with the aim to explain the cause for difference in result. - W3C Prov-Data Model. - Workflow Evolution and its impact on final result. - Graph and Subgraph Isomorphism. - Workflow Differencing using attribute-based graph matching.
c: Chapter Three: Facilitating reproducible research by investigating computational metadata - Introduction - Why-Diff Approach - Provenance Availability. - Use Cases: Twitter Sentiment Analysis using NLTK and DatumBox - YesWorkflow querying the usecase provenance: To understand the difference between two computational result. - NoWorkflow querying the usecase provenance: To understand the difference between two computational result.
d: Chapter Four: Why-Diff: Explaining differences amongst similar workflow runs by exploiting scientific metadata - Introduction - Technology and Standards Used: 1. W3C PROV Data Model. 2. Neo4j 3. eScience Central - Workflow Management System - Why-Diff: 1. Forward Traversal 2. Backward Traversal - Usecases: Simple Variant Interpretation and Freebayes Variant Calling
e: Chapter Five: Analysing Cause and Impact of Reproduced Workflows By Exploiting Execution Provenance. - Introduction - Technical Approach - Graph Matching - Why-Diff: Extension of previous work to accomodate non-isorphic workflows - Visual Representation of Delta Graph with GraphViz - Workflow Comparison - Evaluation and Discussion: this section includes number of experiments for testing "Why-Diff" which has been executed on 20 different workflows and making 90 comparisons considering both isomorphic and non-isomorphic cases of workflow structure.
e: Chapter Six: Explanation of metadata to pinpoint the

differences more precisely. - Evaluation and Results

f: Chapter Seven: Conclusion and Future Works.

IX. Chapter I

The project aim to consider 4 possible cases for reproducing computations. We have used P, X, D, Y to refer to Program, Input, Dependency and Output of the original computation, respectively. Similarly, we denote the same quantities in the reproduced computation by P', X' , D' and Y', respectively. In "Why-Diff", we consider four possible cases for reproducing computations, as follows:

1) $\{P, X, D'\}$ - Changing Dependency keeping Program, input unchanged.
2) $\{ P,X', D\}$ - Changing Input keeping Program, Dependency unchanged
3) $\{P', X, D\}$ - Changing Program keeping Dependency, input unchanged
4) $\{ P', X', D' \}$ - All artefacts changed

Here, we tried to apply first case in which we changed the Dependency keeping Program and Input as constant while redoing computation.

A. *Model usecase for Computational Reproducibility*

Sentiment analysis or opinion mining has emerged as an active domain among the research fraternity because enormous amount of heterogeneous data is continuously increasing every day by the users via www, viz., e-commerce websites, social networks, discussion forums, blogs etc. Motivated by the Natural Language Processing techniques, we have developed a usecase Twitter Sentiment Analysis which is a python script which takes in static tweets and visualise the sentiment score percentage Positive, Negative and Neutral in a pie chart. Lot of Natural Language Processing tools are available to process the human language to classify whether it is subjective or objective and orient its polarity as positive, negative and neutral. TextBlob and NLTK are similar Natural Language Processing tools which provides API which is used to analyse the tweets and compute sentiment score for the text. It helps in converting an unstructured text to structured text which is easily manageable.

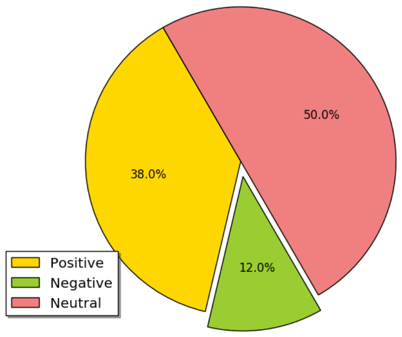


Fig. 1: Sentiment Scores using TextBLOB

While reproducing the usecase, we have intensionally used NLTK (Refer Fig 3) replacing TextBLOB (Refer Fig 2), for the same set of tweets. Positive, Negative and Neutral sentiment percentages using TextBLOB are *38.0%*, *12.0%* and *50.0%* respectively. Whereas NLTK computes *18.0%*, *17.0%* and *65.0%* as Positive, Negative and Neutral percentages respectively. In this case, Y != Y' when trying to mock the case $\{P, X, D'\}$ where D refers TextBLOB and D' refers NLTK respectively.

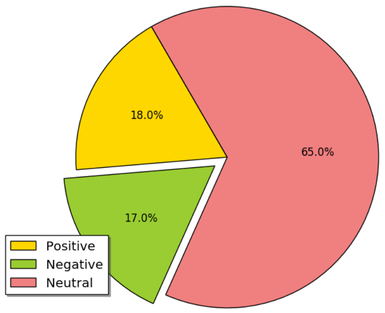


Fig. 2: Sentiment Scores using NLTK

We have observed a striking difference in the results when using different software dependencies keeping the Program and Input unchanged. We viewed TextBLOB and NLTK as a blackbox and recorded the provenance of the two scripts without bothering about the algorithm design of TextBLOB and NLTK. Motivated by the different results from related computations, we have analysed the provenance data collected by YesWorkflow and NoWorkflow. "YesWorkflow" and "noWorkflow" are the tools that captures the provenance of independent scripts. YesWorkflow [31] extracts the latent information from scripts, exports, visualize in graph form. Given a script, YesWorkflow generates the workflow based on user annotations which generally captures prospective provenance. "noWorkflow" [32] (not only Workflow) system uses Python runtime profiling functions to generate provenance traces that reflect the processing history of the script. noWorkflow captures three types of provenance - definition, deployment, and execution provenance.

B. *YesWorkflow*

YesWorkflow is one of the approach to capture prospective provenance from scripts. It helps to interpret the user annotations and visualise the graph that shows the computational steps and data flow in the script.We used YesWorkflow to visualize the graph to extract the API dependency (TextBLOB or NLTK) and retrieved two graphs showing the dependency. Furthermore, it is also possible to visualize the upstream and downstream of a specific data product. In addition to the visualization, YesWorkflow facilitates querying the workflow steps. The user annotations in the script play key role here. This provides very basic prospective provenance which is inadequate for analysing difference between the two execution results.

C. NoWorkflow

Noworkflow is another tool that captures the retrospective provenance automatically. It does not require user annotations for recording provenance. NoWorkflow defines one execution of the experiment as Trial. Here, the execution of Sentiment analysis with TextBLOB is referred as Trial 1 whereas the execution with NLTK referred as Trial 2. “NoWorkflow” has a command called “Diff” which yields a difference graph between 2 executions.

For more details, please visit http://eprint.ncl.ac.uk/file_store/production/234967/D959C504-4859-434A-A59A-8B7DC13AC4BD.pdf

X. Chapter II

The main contribution for the chapter II is the “Why-Diff” algorithm which is able to compare two workflows by “forward” and “backward” traversals.

1. Forward traversal is to find the divergent data nodes that resulted due to change in the artefacts that the researcher voluntarily made self. This aims to traverse the provenance graph of computational steps with experimental input as a starting point. In this case, we aim to provide evidence that the changes have achieved the observed impact.
2. Backward traversal is to find the divergent activity nodes that caused the change that the Researcher unaware of self.This aims to traverse the provenance graph backwards to trace the causes that explain the difference with experimental output as a starting point. By doing so, we aim to find the likely causes for the observed differences in the outputs.

eScience Central, shortly called as “e-SC”, is a Workflow Management System which provides Software and Platform as a Service to scientists to create, edit, run and share Workflows in the cloud [2]. It captures very rich provenance information including data access and Workflow executions. The usage of these provenance details varies from one to another. Some might be interested who created the workflow, who downloaded it, who the data has been shared with. In our perspective, we are looking for what are the workflow inputs, outputs, services and intermediate results.

A. Technologies and standards used

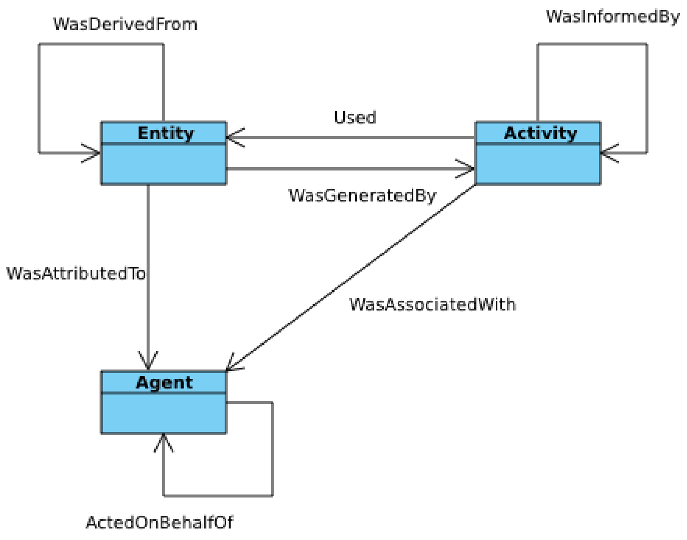


Fig. 3: PROV Core Structures

TABLE I: Provenance Terms Mapping

e-SC Format	PROV Format	Neo4j
Document	Entity	Node
Execution	Activity	Node
used	used	Relationship
wasGeneratedBy	wasGeneratedBy	Relationship

Each input and output files are given identities and file labels. Additionally each service in a particular workflow are given identities as well as service’s configuration are updated. Intermediate files are given hash-values. Having all these piece of data together, we can able to answer why the results of 2 Workflows different and what is the result if a service version is changed say for example from $version1$ to $version2$. In order to verfiy the results of reproduced workflow in accordance with original workflow, an inbuilt differencing system like “Why-Diff” is mandatory. The Workflow execution provenance of e-SC stick to the PROV-DM format. We have collected the Workflow execution provenance of eScience Central and have visualised in form of Neo4j nodes and relationships.

B. Provenance Model

W3C PROV [33] defines a data model and a set of constraints. It represents provenance as a Directed Acyclic Graph. Vertices in the PROV model represents entities, activities or agents. As scientists use different Workflow Management Systems, the provenance generated by such systems are in different format. To address this problem, an integration model ProvONE [34] which takes in the provenance traces of heterogeneous workflow systems and generated integrated provenance database as Prolog facts. To have a common understanding of capabilities of

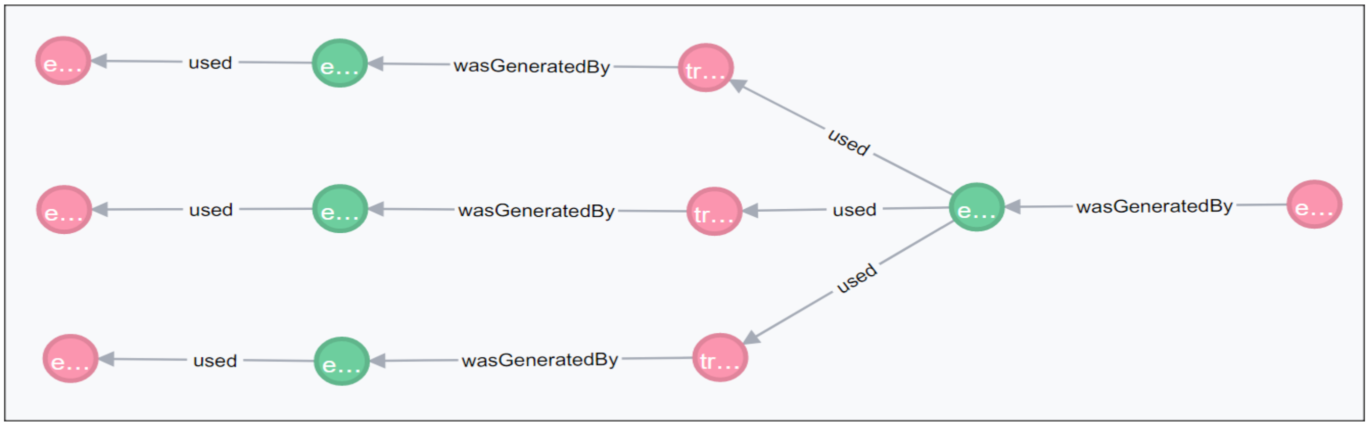


Fig. 4: Neo4j visualization of Sample Workflow

different provenance systems, a data model was crafted by the authors and released as PROV-DM. PROV-DM is conceptual data model that has following nodes and relationships. Nodes are identified by prov:Activity, proc:Entity, prov:Agent, and relationship types are identified by prov:used, prov:wasInformedBy, prov:wasAssociatedWith, prov:wasGeneratedBy, prov:wasAttributedTo, prov:wasDerivedFrom, prov:actedOnBehalfOf. "Entity" refers to Physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary. An "Activity" is something that occurs over a period of time and acts upon or with entities. An "Agent" is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agents activity. "wasGeneratedBy" refers to relationship between entity and activity, saying the entity was generated by the activity. The relationship "used" infers that an entity is used by the activity and "wasDerivedFrom" refers to an entity being derived from another entity. As name indicates,"wasAssociatedWith" refers to an activity being associated to an Agent. "wasInformedBy" infers an activity being informed by another activity about an entity. "actedOnBehalfOf" refers to an agent acting on behalf of another agent. "wasAttributedTo" refers to an entity being attributed to an agent.

C. *Why Neo4j?*

In our project, we have taken advantage of the most popular graph database "Neo4j". The graph database Neo4j is used in majority of fields like Chemistry, social network, semantic web, etc as it enables querying in simple manner in comparison to typical relational database.

In e-SC, the number of available Workflows, services and datasets is constantly increasing. In order to accommodate large voluminous data, the optimal choice would be graph database. Relational database could not handle these datasets as that needs complex join queries. To traverse the high-interconnected Workflow graphs which is combination of entities and activities, Neo4j could be the advantageous. We have analysed Workflow execution provenance of e-SC by exploiting PROV-DM and Neo4j. The Table I represents the provenance terms in e-SC, PROV-DM and Neo4j formats. In the figure 5, all the pink nodes are "entities" whereas green nodes are activities. The flow of graph starts from left to right. Relationships are identified as "used" and "wasGeneratedBy".

D. *WhyDiff*

Why-Diff focuses on the execution provenance of the Workflows rather than the provenance of the Workflows itself. This relies on the execution provenance that are collected by E-Science Central WfMS. As mentioned in our previous work [1], the accuracy of the Why-Diff analysis is proportional to the amount of provenance details of the two Workflows.

The project Why-Diff consider 4 possible cases which resulted in output of reproduced computation different from output of original computation.

1. $\{W, X, D'\}$ - Changing Dependency keeping Workflow, input unchanged.
2. $\{W, X', D\}$ - Changing Input keeping Workflow, Dependency unchanged.
3. $\{W', X, D\}$ - Changing Workflow keeping Dependency, input unchanged.
4. $\{W', X', D'\}$ - All artefacts changed.

We consider the difference in the Workflow results may be intentional or unintentional. Intentional changes are something that the experimenter is making himself/herself whereas unintentional changes are the changes that is made without the knowledge of the experimenter (For example updated external dependencies or external data sources which is not actually in control of the experimenter). In our previous work [1], We have considered above cases and proved that there is possibility of the output difference when there is no change in Inputs X and Program P changing the Dependency D to D', mocking the scenario $\{P, X, D'\}$. The same applies to Workflows as well except the fact the Program P represents orchestration of Program statements where Workflow W represents orchestration of services. While tracking provenance of independent scripts, it is essential to keep track of each data product produced or consumed out of a particular program

ALGORITHM 1: Forward Traversal

```
Input: Invocation1_Id
Input: Invocation2_Id
Output: Δ Graph
$Compare\ the\ list\ of\ first\ Entities\ in\ both\ graphs$;
$first\_hop = true$;
do
    if $first\_hop == true$ then
        $List < Activity > Activity1List \leftarrow findFirstActivityList(Invocation1\_Id)$;
        $List < Activity > Activity2List \leftarrow findFirstActivityList(Invocation2\_Id)$;
    else
        $List < Activity > Activity1List \leftarrow firstActivityNewList$;
        $List < Activity > Activity2List \leftarrow secondActivityNewList$;
        $Clear firstActivityNewList$ & $secondActivityNewList$;
    end
    foreach $(A1, A2) \in zip(Activity1List, Activity2List)$ do
        $List < Entity > Entity1List \leftarrow getWasGeneratedBy(A1)$;
        $List < Entity > Entity2List \leftarrow getWasGeneratedBy(A2)$;
        foreach $(E1, E2) \in zip(Entity1List, Entity2List)$ do
            $\Delta Graph \leftarrow Compare(E1, E2)$;
            $List < Activity > returnedActivityList1 \leftarrow getUsed(E1)$;
            $List < Activity > returnedActivityList2 \leftarrow getUsed(E2)$;
            foreach $(A1, A2) \in zip(returnedActivityList1, returnedActivityList2)$ do
                $Add\ A1\ to\ firstActivityNewList$;
                $Add\ A2\ to\ secondActivityNewList$;
            end
        end
    end
while $firstActivityNewList \neq Empty$ && $secondActivityNewList \neq Empty$;
$Compare\ the\ list\ of\ last\ Entities\ in\ both\ graphs$;
Function Compare (E1, E2)
    if $E1.fileLabel \neq E2.fileLabel$ then
        $\Delta Graph \leftarrow (E1, E2)$;
    end
    return $\Delta Graph$
Function findFirstActivityList (InvocationId)
    $List\ all\ activityNodes\ WHERE\ activity - [USED] - entity\ WHERE\ NOT\ entity - [WASGENERATEDBY] - any\_activity\ AND\ activity.id\ CONTAINS\ InvocationId$;
    return activityNodes
```

statement. In contrast to the scripts, workflows express the computations as service-level components. It considers each computational block as a service, which makes it feasible to compare two workflows based on the service version. It is more convenient to visually relate the Workflow steps which consumes data and produce data in the form of graph. At this early stage of our investigation, we make the simplifying assumption that the provenance graphs for the two executions are isomorphic, and only focus on detecting differences in properties of corresponding nodes. We use the retrospective provenance of the Workflow but not the abstract provenance of the Workflow. The Workflows to be compared are structurally similar (i.e. isomorphic). This is the simplest possible scenario where the topologies of the two graphs are identical, and only the properties on the nodes differ from one graph to another.

1) Forward Traversal: Algorithm I is designed to move forward the provenance graph to detect diverging entity nodes because of an intentional change introduced by the experimenter himself. If the changes are introduced intentionally, then the experimenter would like to know the effect (data difference that is resulted) of the change. Hence he/she forward traverse the graph to identify the data nodes that are not similar in both the Workflow graphs. We consider this as "effect-analysis". In the algorithm, we have used symbols $E1$, $E2$, $A1$, $A2$ to refer entity node for first invocation , entity node for second invocation, activity node for first invocation, activity node for second invocation respectively. The start nodes of the traversal are picked by listing the activities which have used an entity which is not generated by any other activity. As the traversal proceeds, for each activity, the generated entities are compared. The traversal continues till generated entities have not been used by any further activity. The traversal compares the file label (or hash value) initially. In eScience central WfMS, the end nodes are attached with file label property which is used to find the physical file name. The intermediate (we refer as transient) files possess the "hash value" property which is the unique identity of the transient data. All the diverging entities are added to the Δ Graph. The zip function in the algorithm ensures that the nodes from graph$_1$ are compared with the corresponding nodes in graph$_2$.

2) Backward Traversal: Algorithm II is designed to move backward the provenance graph to detect diverging activity nodes because of an unintentional change which the experimenter does not know. If the changes are unintentional then the experimenter would like to know the cause (activity node that caused the difference). And so he/she backward traverse the graph to identify the activity nodes that are not similar in both the Workflow graphs. This can be termed as "cause-analysis". In the algorithm, we have used symbols $E1$, $E2$, $A1$, $A2$ to refer entity node for first invocation , entity node for second invocation, activity node for first invocation, activity node for second invocation respectively. The start nodes of the traversal are picked by listing the activities which have "generated" an entity which is further not "used" by any other activity. As the traversal proceeds, each activity configuration are compared and for each activity the "used" entities are identified. The

ALGORITHM 2: Backward Traversal

```
Input: Invocation1_Id
Input: Invocation2_Id
Output: Δ Graph
$first\_hop = true$;
do
    if $first\_hop == true$ then
        $List < Activity > Activity1List \leftarrow findLastActivityList(Invocation1\_Id)$;
        $List < Activity > Activity2List \leftarrow findLasttActivityList(Invocation2\_Id)$;
    else
        $List < Activity > Activity1List \leftarrow firstActivityNewList$;
        $List < Activity > Activity2List \leftarrow secondActivityNewList$;
        $Clear firstActivityNewList$ & $secondActivityNewList$;
    end
    foreach $(A1, A2) \in zip(Activity1List, Activity2List)$ do
        $\Delta Graph \leftarrow Compare(A1, A2)$;
        $List < Entity > Entity1List \leftarrow getUsed(A1)$;
        $List < Entity > Entity2List \leftarrow getUsed(A2)$;
        foreach $(E1, E2) \in zip(Entity1List, Entity2List)$ do
            $List < Activity > returnedActivityList1 \leftarrow getWasGeneratedBy(E1)$;
            $List < Activity > returnedActivityList2 \leftarrow getWasGeneratedBy(E2)$;
            foreach $(A1, A2) \in zip(returnedActivityList1, returnedActivityList2)$ do
                $\Delta Graph \leftarrow Compare(A1, A2)$;
                $Add\ A1\ to\ firstActivityNewList$;
                $Add\ A2\ to\ secondActivityNewList$;
            end
        end
    end
while $firstActivityNewList \neq Empty$ && $secondActivityNewList \neq Empty$;
Function Compare (A1, A2)
    if $A1.configuration \neq A2.configuration$ then
        $\Delta Graph \leftarrow (A1, A2)$;
    end
    return $\Delta Graph$
Function findLastActivityList(InvocationId)
    $List\ all\ activityNodes\ WHERE\ entity - [WASGENERATEDBY] - activity\ WHERE\ NOT\ any\_activity - [USED] - entity\ AND\ activity.id\ CONTAINS\ InvocationId$;
    return activityNodes
```

traversal continues till "used" entities have not been generated by any further activity. In eScience central WfMS, each activity are associated with configuration property, which captures good provenance of which the interesting ones are the file labels it has used/ generated, library used, the conditions it has used, source file used, block name and version. These metadata will help us comparing one activity node to another and all the diverging activities are added to the Δ Graph. The zip function in the algorithm ensures that the nodes from graph$_1$ are compared with the corresponding nodes in graph$_2$.

XI. Chapter III

The main contribution is a tool "Why-Diff" which is able to analyse the differences between any two workflow runs by exploiting the execution provenance associated with the workflow.

1. Possible Variations between Original and Reproduced Computations: As we already introduced the notations W, X, D as Workflow, Input and Dependencies and variation to these could affect the final result. Variation to the input i.e. X could be X' and D could be D' which can be introduced in reproduced workflows by changing the input entities in the workflow. However variation to the workflow i.e. W to W' actually mean a new task has been added to the workflow, an existing task is deleted from the workflow or updation of existing task. To mitigate this, we come up with real-time cases of workflow evolution.
   a. Addition of new activities.
   b. Deletion of existing activities.
   c. Updated version of existing activities.

   For example consider an Workflow W which consists of workflow blocks [$w1$, $w2$, $w3$, $w4$, $w5$]. In the reproduced workflow W', there could be an insertion of $w6$ i.e [$w1$, $w2$, $w3$, $w4$, $w5$, $w6$], or deletion of $w2$ i.e. [$w1$, $w3$, $w4$, $w5$] or updating an existing activity $w4$ i.e. [$w1$, $w2$, $w3$, $w4'$, $w5$] or combination of all these. Here the addition and deletion of workflow blocks give raise to the non-isomorphic cases where a workflow node is not present in the other graph.
2. Why-Diff - an extension of our previous algorithm [30] is to find the divergent activity and data nodes that resulted to support non-isomorpic cases. This aims to traverse the provenance graph of workflow to trace the causes that explain the difference. By doing so, we aim to find the causes for the observed differences in the outputs.
3. Content-based differencing between entities of the comparing workflows.
4. Δ graph - a GraphViz implementation of graph visualisation, comprising of matching nodes from comparing workflow, and highlighting the unmatching nodes with subgraph.

A. Technical Approach

Figue 1(a) and 1(b) represents the insertion/deletion of activity respectively. The model that we exhibit here follows PROV-DM [33], where all data produced/consumed is an "entity" (oval-shaped nodes) and the workflow blocks which uses/generates an entity is an "activity" (box-shaped nodes). The "used" relationships are represented in blue arrows and the "wasGeneratedBy" relationships are represented in brown arrows. The orange-dashed lines link the matching/corresponding nodes. By terming it as corresponding activity, we clarify that the activity node in first graph may or may not be equal as the activity node in the second graph, however, they both are similar. And green dashed line links "sync" nodes which is the first matching node after the divergence.

1. Activity Insertion: This refers to the insertion of a single workflow block to the reproduced workflow. Fig:1.a represents the sample activity insertion where $E3$ is generated because of the insertion of new Activity $A2$.
2. Activity Deletion: This refers to the deletion of a single workflow block to the reproduced workflow. Fig:2.b represents that an activity $A2$ is removed in second workflow and hence there is no existence of $E3$.
2. Activity Update: This refers to the update of a single workflow block to the reproduced workflow. Fig:3.b represents that an activity $A3$ is replaced in second workflow with $E3'$.

B. Why-Diff

Why-Diff focuses on the execution provenance of the workflows rather than the provenance of the workflows itself. This relies on the execution provenance that is collected by WfMS. As mentioned in our previous work, the accuracy of the Why-Diff analysis depends on the amount of provenance details of the two workflows. The algorithm uses symbols $E1$, $E2$, $A1$, $A2$ to refer entity node for 1st invocation , entity node for 2nd invocation, activity node for 1st invocation, activity node for 1nd invocation respectively. Algorithm is an extension of our previous work [30] which aims to find the divergent nodes so that know the cause for the different results. This compares two graphs which is not structurally similar.

To start the traversal, the researcher should know the invocation ID of the two workflows that he/she wants to compare. The traversal starts by listing the activities which may have "used" an entity say E, which was "not generated" by any other activity. An activity can be uniquely identified using activity's id. Hence, the listed activites are zipped using "Zip" function which is to arrange the activities from first graph and second graph based on its ids. This is to ensure the activities of the first graph is actually compared against the corresponding activity in the second graph. All the activities which is not zipped are added to the $List < unZippedNode >$ nodes. The 'zipped' activites from first workflow are compared against the 'zipped' activities of second workflow. The comparion is made to check whether the configuration of activity from first graph is equal to the configuration of activity from second graph. Some of the activity's configuration details that eScience Central Workflow Management Systems captures include AllowRetriesOnTimeOut, Condition used, DebugMode, DebugPort, DebugSuspended, Delimiters, EnforeInvocationTimeOut, ImportChunkSize, InvocationTimeout, Label, Name, Source, ProceedOnFail, TimeOutRetries. As the traversal proceeds, for each activity, the corresponding "generated" entities (zipped) are compared. Unzipped entities are added to the $List < unZippedNodes >$. The traversal stops when it reaches last node (i.e final output) in the workflow graph. If the activity node in the first workflow is not equal to the corresponding activity node in the second workflow, then those nodes are added to the list of diverging nodes which aids in representing difference graph. (i.e. Δ Graph).

In eScience central WfMS, the entity nodes are attached to file label property which is used to find the physical file name. So, we compare the contents of these nodes. The intermediate (we refer as transient) files possess the "hash value" property which helps to identify unique transient data. For these intermediate nodes, we comapre hash-values. If the file label or the hash value of entity node in the first workflow is not equal to the file label or hash value of its corresponding entity node in the second workflow, then those nodes are added to the list of diverging nodes which aids in representing difference graph. (i.e. Δ Graph). To handle non-isomorphic cases, all the extra nodes are added to the $List < divergingNodes >$. Further, we keep track of all unzipped nodes during the traversal and try to resync each unzipped node with the every node in the $List < divergingNodes >$ to find the sync node. The sync nodes are compared and the diverging node list is updated accordingly. We use the $List < divergingNodes >$ and $List < zippedNodes >$ to construct the delta graph.

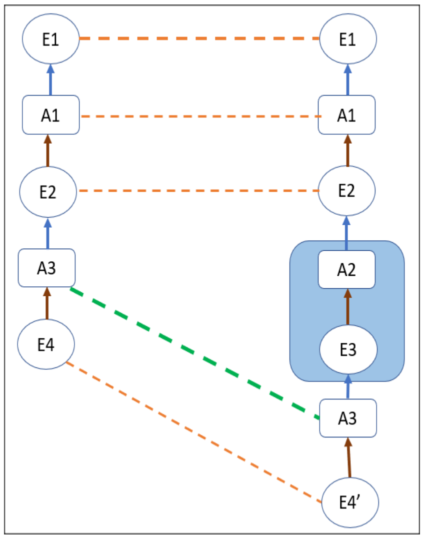
C. Visual Representation of Delta Graph with GraphViz

To visually represent the delta graph, we make use of GraphViz. GraphViz is a open-source software, which facilitates to draw graphs specified in DOT [35]. All the entities are represented in pink-oval shaped nodes and all the activities are represented in green-box shaped nodes. All the entity names of first and second graphs are represented by 'e' with subscript 1 and 2 respectively, followed by the node number. All the activity names of first and second graphs are represented by 'a' with subscript 1 and 2 respectively, followed by the node number. For example, '$e_1 2$' refers to the 2nd entity of first graph and '$a_2 3$' refers to the 3rd activity of second graph. If the entity/activity from 1st graph is matched/zipped with node in 2nd graph, it is represented with '$\equiv$' equivalent symbol. For example, '$e_1 2 \equiv e_2 2$' indicates 2nd entity from 1st graph is equivalent to 2nd entity from 2nd graph. Subgraphs specify semantic information about graph components. GraphViz provides facility to group certain nodes and edges in the form of subgraph which we made use of highlighting the unmatched/unzipped nodes as well as unmatched edges. To highlight the unequal nodes, the node attribute "peripheries" is set to 2 so that to have two borders for a node.

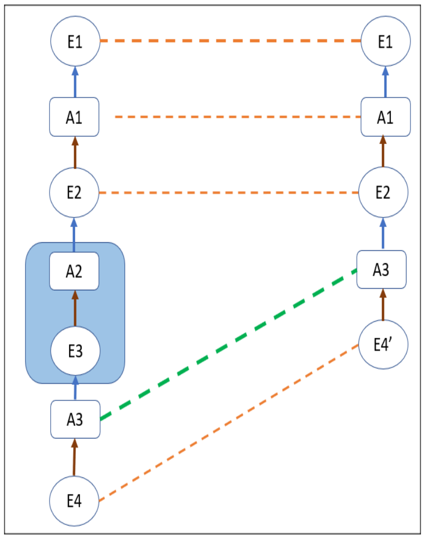
XII. Research Outputs

A. Usecase - Freebayes Variant Calling

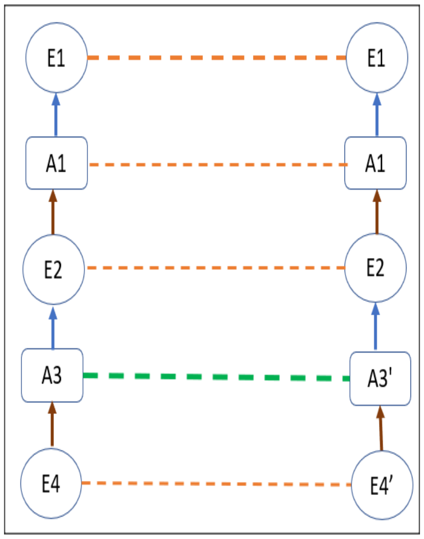
The use case we worked with was the Freebayes variant calling workflow. It processes aligned patient sequence



(a) Activity Insertion (Non-isomorphic)

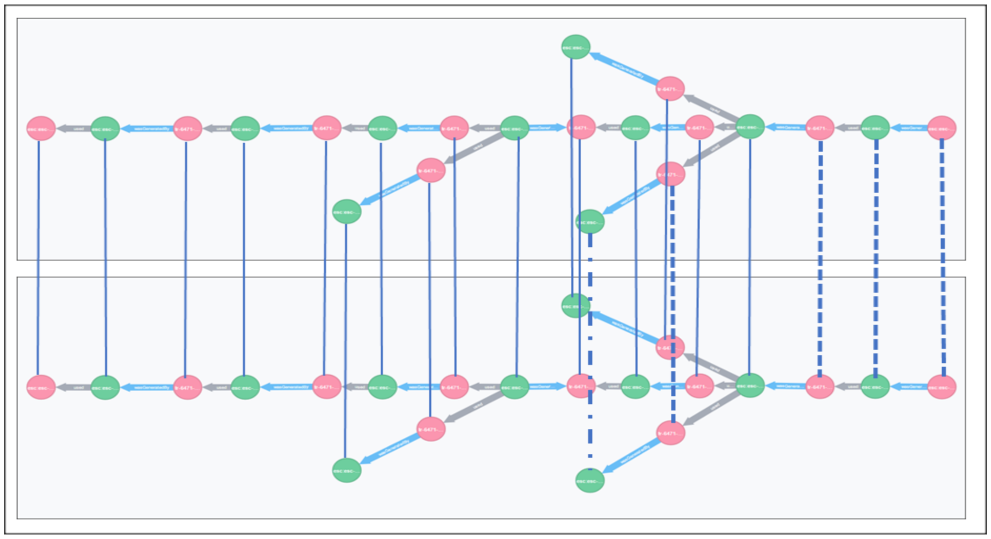


(b) Activity Deletion (Non-isomorphic)

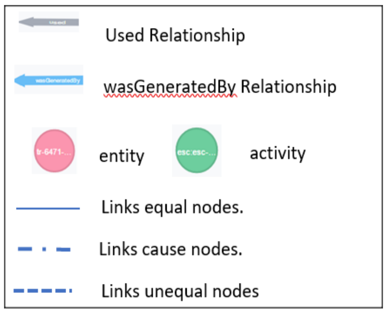


(c) Activity Update (Isomorphic)

Fig. 5: Activity Insertion, Deletion and Update



(a) Difference Graph of Freebayes



(b) Notations

Fig. 6: Comparison graph of Freebayes and notations used

files (in the BAM format) in the search for variants from the reference genome. The main source of change in this workflow is the version of Freebayes tool used (https://github.com/ekg/freebayes). Over the last three years the author has released about 15 versions of the program and so an important question is whether and to what extent these changes affect the output.

1) Evaluation of Freebayes Workflow: In fig 6, we have visualised the graphs of the two Freebayes invocations of which the top graph represents invovation 6471 whereas the bottom graph represents invocation 5463. The difference between the invocations are such that 6471 has used Freebayes library of version $Freebayes - 1.0.2$ whereas 5463 has used $Freebayes - 0.9.10$. To spot similar, dissimilar and cause nodes, we have used three types of lines. The straight line refers the two linked nodes are equal. “Dashed” lines link the nodes that are not equal. “Dashed dot” lines link the nodes that actually caused the difference.

We have tested the algorithm with another usecase “Simple Variant Interpretation (SVI)”, which was implemented as part of Re-Comp project (recomp.org.uk). SVI is a workflow which takes advantage of two external data sources OMIM GeneMap and NCBI ClinVar to provide interpretation of human variants to facilitate clinical diagnosis of genetic diseases say for example, Alzheimer’s disease. We have observed above stated Why-Diff scenario in SVI Workflow. For more results on SVI, please visit https://github.com/Priyaa140624191/Why-Diff/blob/master/WhyDiffAlgorithm/SVI%20Results.pdf

ALGORITHM 3: Why-Diff

```
Input: Invocation1_Id,Invocation2_Id
Output: $\Delta$ Graph
Begin
    $Compare\ the\ list\ of\ first\ \&\ last\ Entities\ in\ both\ graphs$;
    $first\_hop = true$;
    do
        if $first\_hop == true$ then
            $List < Activity > Activity1List \leftarrow$
              $findFirstActivityList(Invocation1\_Id)$;
            $List < Activity > Activity2List \leftarrow$
              $findFirstActivityList(Invocation2\_Id)$;
        else
            $List < Activity > Activity1List \leftarrow$
              $firstActivityNewList$;
            $List < Activity > Activity2List \leftarrow$
              $secondActivityNewList$;
            $Clear firstActivityNewList$ &
              $secondActivityNewList$;
        end
        foreach $(A1) \in firstActivityNewList$ do
            foreach $(A2) \in secondActivityNewList$ do
                if $A1 \xi A2$ then
                    $List < zippedNodes > \leftarrow Add(A1, A2)$;
                    $List < divergingNodes > \leftarrow$
                      $Compare(A1, A2)$;
                end
                $AddUnMatchedNodes(A1, A2)$;
                $List < Entity > Entity1List \leftarrow$
                $getWasGeneratedBy(A1)$;
                $List < Entity > Entity2List \leftarrow$
                  $getWasGeneratedBy(A2)$;
                foreach $(E1) \in Entity1List$ do
                    foreach $(E2) \in Entity2List$ do
                        $List < firstGraph > \leftarrow Add(A1, \leftarrow$
                          $, E1)$;
                        $List < secondGraph > \leftarrow Add(A2, \leftarrow$
                          $, E2)$;
                        if $E1 \xi E2$ then
                            $List < zippedNodes > \leftarrow$
                              $Add(E1, E2)$;
                              $List < divergingNodes > \leftarrow$
                              $Compare(E1, E2)$;
                        end
                        $addUnMatchedNodes(E1, E2)$;
                        $List < Activity >$
                        $returnedActivityList1 \leftarrow getUsed(E1)$;
                        $List < Activity >$
                          $returnedActivityList2 \leftarrow$
                          $getUsed(E2)$;
                        foreach $(A1) \in returnedActivityList1$ do
                            foreach
                              $(A2) \in returnedActivityList2$ do
                                $List < firstGraph > \leftarrow$
                                  $Add(E1, \leftarrow, A1)$;
                                $List < secondGraph > \leftarrow$
                                  $Add(E2, \leftarrow, A2)$;
                                $List <$
                                  $firstActivityNewList > \leftarrow$
                                  $Add(A1)$;
                                $List <$
                                  $secondActivityNewList > \leftarrow$
                                  $Add(A2)$;
                            end
                        end
                        if $returnedActivityList1 \neq$
                          $Empty \&\& returnedActivityList2 =$
                          $Empty$ then
                            $addExtraNodesFirstGraph(returnedActivityList1)$;
                        end
                        if $returnedActivityList1 =$
                          $Empty \&\& returnedActivityList2 \neq$
                          $Empty$ then
                            $addExtraNodesSecondGraph(returnedActivityList2)$;
                        end
                    end
                end
            end
        end
    while $firstActivityNewList \neq Empty\ \&\&$
      $secondActivityNewList \neq Empty$;
    $resync()$;
    $constructDeltaGraph$(List<firstGraph>,List<secondGraph>,List<divergingNodes>,List<zippedNodes>);

    Function Compare (E1, E2)
        if $E1 \neq E2$ then
            $List < divergingNodes > <- Add(E1, E2)$;
        end
        return $List < divergingNodes >$;
```

ALGORITHM 4: Why-Diff

```
    Function resync()
        foreach $(Node1) \in List < firstExtra >$ do
            foreach $(Node2) \in List < secondExtra >$ do
                if $Node1\ is\ a\ Activity\ \&\&\ Node2\ is\ a\ Activity$
                  then
                    if $Node1 \xi Node2$ then
                        $List < divergingNodes > \leftarrow$
                          $Compare(A1, A2)$;

                if $Node1 \xi Node2$ then
                    $List < divergingNodes > \leftarrow$
                      $Compare(E1, E2)$;

    Function addUnMatchedNodes(Node1, Node2)
        $List < firstExtra > \leftarrow Add(Node1)$;
          $List < secondExtra > \leftarrow Add(Node2)$;
          $List < divergingNodes > \leftarrow Add(Node1, ABSENT)$;
          $List < divergingNodes > \leftarrow Add(ABSENT, Node2)$;
    Function addExtraNodesFirstGraph( returnedActivityList1)
        do
            foreach $(A1) \in returnedActivityList1$ do
                $List < divergingNodes > \leftarrow$
                  $Add(Node1, ABSENT)$;
                $List < firstExtra > \leftarrow Add(Node1)$;
                $List < Entity > Entity1List \leftarrow$
                  $getWasGeneratedBy(A1)$;
                foreach $(E1) \in Entity1List$ do
                    $List < divergingNodes > \leftarrow$
                      $Add(Node1, ABSENT)$;
                    $List < firstExtra > \leftarrow Add(Node1)$;
                    $List < Activity >$
                      $returnedActivityList1 \leftarrow getUsed(E1)$;

        while $returnedActivityList1\ has\ Next$;
    Function addExtraNodesSecondGraph( returnedActivityList2)
        do
            foreach $(A1) \in returnedActivityList1$ do
                $List < divergingNodes > \leftarrow$
                  $Add(ABSENT, Node2)$;
                $List < secondExtra > \leftarrow Add(Node2)$;
                $List < Entity > Entity1List \leftarrow$
                  $getWasGeneratedBy(A1)$;
                foreach $(E1) \in Entity1List$ do
                    $List < secondExtra > \leftarrow Add(Node2)$;
                    $List < divergingNodes > \leftarrow$
                      $Add(ABSENT, Node1)$;
                    $List < Activity >$
                      $returnedActivityList1 \leftarrow getUsed(E1)$;

        while $returnedActivityList1\ has\ Next$;
    Function findFirstActivityList (InvocationId)
        $List\ all\ activityNodes\ WHERE$
          $activity - [USED] - entity\ WHERE\ NOT$
          $entity - [WASGENERATEDBY] - any\_activity$
          $AND\ activity.id\ CONTAINS\ InvocationId$;
        return activityNodes
    Function Compare (A1, A2)
        if $A1.configuration \neq A2.configuration$ then
            $List < divergingNodes > <- Add(A1, A2)$;

        return $List < divergingNodes >$;
    Function constructDeltaGraph($List < firstGraph >$,$List <$
      $secondGraph >$,$List < divergingNodes >$,$List <$
      $zippedNodes >$)
        foreach $row \in List < firstGraph >$ do
            $\Delta Graph < -AddAll$(row);

        foreach $row \in List < secondGraph >$ do
            $\Delta Graph < -AddAll$(row);

        foreach $Node1 \in List < zippedNodes >$ &&$Node2 \in$
          $List < zippedNodes >$ do
            $\Delta Graph < -Edit$(Node1,Node1≡Node2);
            $\Delta Graph < -Edit$(Node2,Node1≡Node2);

        foreach $Node1 \notin List < zippedNodes >$|| $Node2 \notin List <$
          $zippedNodes >$ do
            $SubGraph < -AddAll$(Node1,Node2);

        foreach $roNodew \in List < divergingNodes >$ do
            $\Delta Graph < -Edit(Node, peripheries = "2", style =$
              $"filled", fillcolor =$
              $"mediumspringgreen", color = "orangered")$;

        return $\Delta Graph$;
```

B. *Workflow Comparison supporting Non-Isomorphic Cases*

The figure Fig.8(a) shows the comparison between two workflows, in which activity '$a_2$1' of first workflow has been inserted serially in reproduced computation. The resultant Δ Graph shows inserted nodes from the first workflow as subgraph. The '$a_1$2$\equiv$$a_2$1' refers '$a_1$2' is equivalent to '$a_1$1' but they are "unequal" as the node is highlighted with double-red borders. Hence, the cause for the difference in results (i.e. '$e_1$3' is not equal to 'e$_2$3') is because activity '$a_1$2' is replaced with activity '$a_2$1' and the activity '$a_2$1' is inserted. The figure Fig.8(b) shows activity deletion comparing same two workflows as Fig.8(a) by changing the order, thus the subgraph highlights the nodes deleted serially.

The figure Fig.9(a) shows the comparison between two workflows, in which path comprising of activities '$a_2$2' and '$a_2$4' of first workflow has been inserted parallely in reproduced computation. The resultant Δ Graph shows inserted nodes from the first workflow as subgraph. The '$a_1$5$\equiv$$a_2$3' refers '$a_1$5' is equivalent to '$a_1$3' but they are "unequal" as the node is highlighted with double-red borders. Hence, the cause for the difference in results (i.e. '$e_1$6' is not equal to e$_2$4') is because activity '$a_1$5' is replaced with activity '$a_2$3' and the activities running parallely to existing activites'$a_2$2' and '$a_2$4' are removed. The figure Fig.9(b) shows path deletion comparing same two workflows as Fig.9(a) by changing the order, thus the subgraph highlights the path (comprising of nodes), deleted parallely.

The figure Fig.10(a) shows the comparison between two workflows, in which sub-workflow comprising of activities '$a_2$1', '$a_2$2', '$a_2$3' and '$a_2$4' of first workflow has been inserted in reproduced computation. This is complex scenario which is a combination of serial and parallel activity insertion. The resultant Δ Graph shows inserted nodes from the first workflow as subgraph. The '$a_1$5$\equiv$$a_2$1' refers '$a_1$5' is equivalent to '$a_1$1' but they are "unequal" as the node is highlighted with double-red borders. Hence, the cause for the difference in results (i.e. '$e_1$6' is not equal to 'e$_2$2') is because activity '$a_1$5' is replaced with activity '$a_2$1' and the subworkflow (running serially to the existing activities) comparising of '$a_2$1', '$a_2$2', '$a_2$3' and '$a_2$4' is removed. The figure Fig.10(b) shows subworkflow deletion comparing same two workflows as Fig.10(a) by changing the order, thus the subgraph highlights the subworkflow (comprising of nodes) deleted.

C. *Evaluation*

We have tested 'Why-Diff' with 10 similar workflows taking multiple variations into consideration, trying out all possible combinations changing artefacts W, X, Y to W', X', Y' as well as testing non-isomorphic cases (i.e. activity, path and subworkflow insertion/deletion). We have executed 45 comparisons among the 10 workflows. Figure.11 represents workflow comparisons of 140104 with 9 similar workflows (i.e. 140102, 140103, 140105, 140109, 140110, 140100, 140101, 140022, 140033). Number of matched, unmatched, unequal, total nodes in Δ Graph in each comparisons are represented in bar chart to figure out how the final results vary when inserting and/or deleting an activity while reproducing a workflow.

References

[1] P. Thavasimani and P. Missier, "Facilitating reproducible research by investigating computational metadata," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3045–3051.

[2] H. Hiden, S. Woodman, P. Watson, and J. Cala, "Developing cloud applications using the e-science central platform," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1983, 2012. [Online]. Available: http://rsta.royalsocietypublishing.org/content/371/1983/20120085

[3] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar, "Provenance trails in the wings/pegasus system," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 587–597, 2008.

[4] E. Afgan, D. Baker, M. van den Beek, D. Blankenberg, D. Bouvier, M. ech, J. Chilton, D. Clements, N. Coraor, C. Eberhard, B. Grning, A. Guerler, J. Hillman-Jackson, G. Von Kuster, E. Rasche, N. Soranzo, N. Turaga, J. Taylor, A. Nekrutenko, and J. Goecks, "The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update," *Nucleic Acids Research*, vol. 44, no. W1, p. W3, 2016. [Online]. Available: +http://dx.doi.org/10.1093/nar/gkw343

[5] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Visualization meets data management," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 745–747. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142574

[6] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, Aug. 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.v18:10

[7] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: Lessons in creating a workflow environment for the life sciences: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1067–1100, Aug. 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.v18:10

[8] M. Schwab, N. Karrenbach, and J. Claerbout, "Making scientific computations reproducible," *Computing in Science & Engineering*, vol. 2, no. 6, pp. 61–67, 2000.

[9] N. Barnes, "Publish your computer code: it is good enough," *Nature*, vol. 467, no. 7317, pp. 753–753, 2010.

[10] J. P. Mesirov, "Accessible Reproducible Research," vol. 327, no. January, pp. 415–417, 2010.

[11] A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz, "Research priorities. Shining light into black boxes." *Science (New York, N.Y.)*, vol. 336, no. 6078, pp. 159–60, 2012.

[12] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best Practices for Scientific Computing," *PLoS Biology*, vol. 12, no. 1, 2014.

[13] J. Vitek and T. Kalibera, "Repeatability, Reproducibility, and Rigor in Systems Research," *Proceedings of the ninth ACM international conference on Embedded software - EMSOFT '11*, pp. 33–38, 2011.

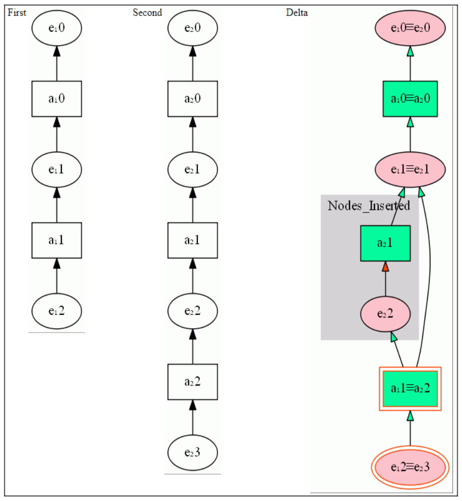
[14] D. C. Drummond, "Replicability is not reproducibility: Nor is it good science," *Proceedings of the Evaluation Methods for Machine Learning Workshop 26th International Conference for Machine Learning*, no. 2005, pp. 1–4, 2009.

[15] D. G. Feitelson, "From repeatability to reproducibility and corroboration," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 3–11, 2015.

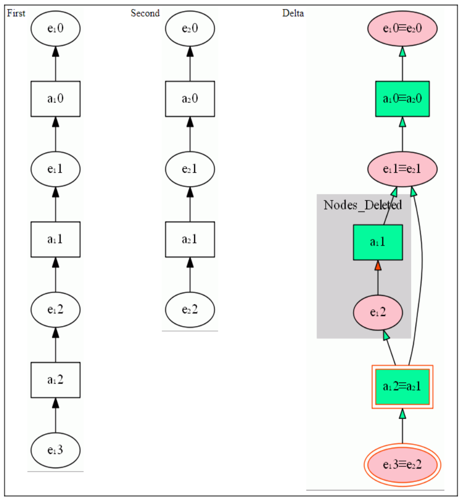
[16] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva, "Bridging workflow and data provenance using strong links," in *International Conference on Scientific and Statistical Database Management*. Springer, 2010, pp. 397–415.

[17] ——, "Bridging workflow and data provenance using strong links," in *International Conference on Scientific and Statistical Database Management*. Springer, 2010, pp. 397–415.

[18] T. U. P. Granularities, "Provenance and annotation of data and processes," *Lecture Notes in Computer Science*, vol. 7525, 2012.

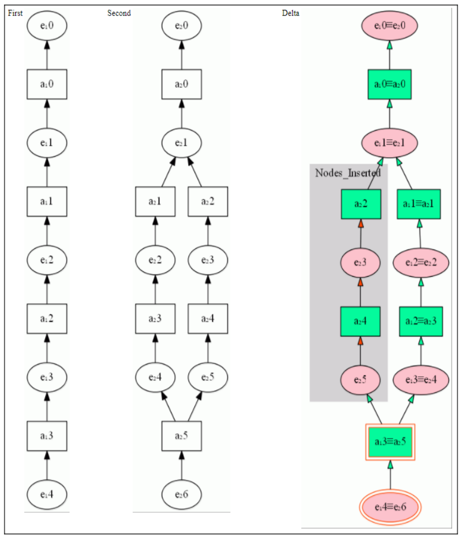


(a) Activity Insertion

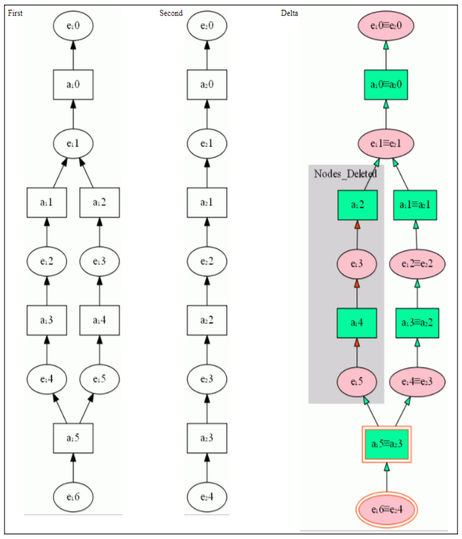


(b) Activity Deletion

Fig. 7: Activity Insertion and Deletion



(a) Path Insertion



(b) Path Deletion

Fig. 8: Path Insertion and Deletion

[19] T. Miksa, R. Mayer, and A. Rauber, "Ensuring sustainability of web services dependent processes," *International Journal of Computational Science and Engineering*, vol. 10, no. 1-2, pp. 70–81, 2015.

[20] D. McGuinness, T. Lebo, and S. Sahoo, "PROV-o: The PROV ontology," W3C, W3C Recommendation, Apr. 2013. [Online]. Available: http://www.w3.org/TR/prov-o/

[21] T. Miksa, S. Pröll, R. Mayer, S. Strodl, R. Vieira, J. Barateiro, and A. Rauber, "Framework for verification of preserved and redeployed processes," 2013.

[22] A. Davison, "Automated capture of experiment context for easier reproducibility in computational research," *Computing in Science & Engineering*, vol. 14, no. 4, pp. 48–56, 2012.

[23] J. a. F. Pimentel, J. Freire, V. Braganholo, and L. Murta, "Tracking and analyzing the evolution of provenance from scripts," in *Proceedings of the 6th International Provenance and Annotation Workshop on Provenance and Annotation of Data and Processes - Volume 9672*, ser. IPAW 2016. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 16–28. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-40593-3_2

[24] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo, "Managing rapidly-evolving scientific workflows," in *Provenance and Annotation of Data*, L. Moreau and I. Foster, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 10–18.

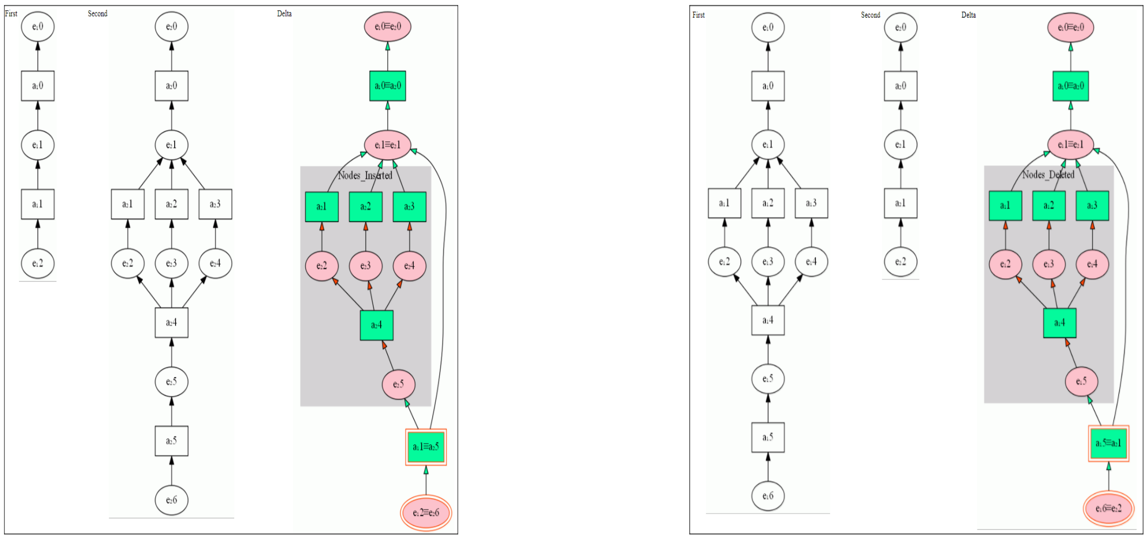
[25] Z. Bao, S. Cohen-Boulakia, S. B. Davidson, and P. Girard, "Pdiffview: Viewing the difference in provenance of workflow results," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1638–1641, Aug. 2009. [Online]. Available: https://doi.org/10.14778/1687553.1687612

[26] Z. Bao, S. Cohen-Boulakia, S. B. Davidson, A. Eyal, and S. Khanna, "Differencing provenance in scientific workflows," in *2009 IEEE 25th International Conference on Data Engineering*, March 2009, pp. 808–819.

[27] N. Kong, T. Grossman, B. Hartmann, M. Agrawala, and G. Fitzmaurice, "Delta: A tool for representing and comparing workflows," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012, pp. 1027–1036. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208549

[28] P. Buneman and S. Staworko, "Rdf graph alignment with bisimulation," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1149–1160, Aug. 2016. [Online]. Available: http://dx.doi.org/10.14778/2994509.2994531

[29] F. Depiero, M. Trivedi, and S. Serbin, "Graph matching using



(a) Sub-Workflow Insertion

(b) Sub-Workflow Deletion

Fig. 9: Sub-Workflow Insertion and Deletion

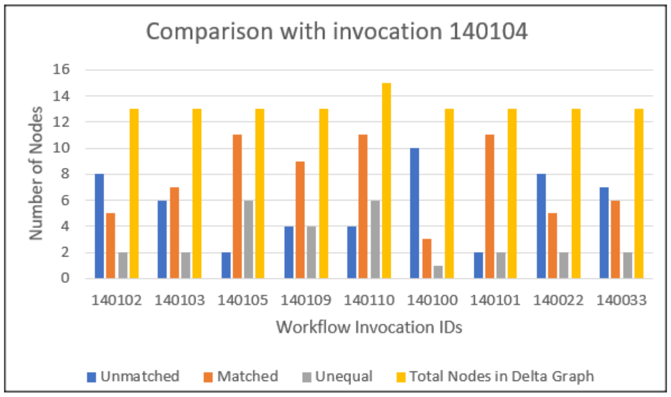


Fig. 10: Workflow Comparisons

a direct classification of node attendance," *Pattern Recognition*, vol. 29, no. 6, pp. 1031 – 1048, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031320395001409

[30] P. Thavasimani, J. Caa, and P. Missier, "Why-diff: Explaining differences amongst similar workflow runs by exploiting scientific metadata," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, pp. 3031–3041.

[31] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S. Dey, J. Freire, D. Huntzinger, C. Jones, D. Koop, P. Missier, M. Schildhauer, C. Schwalm, Y. Wei, J. Cheney, M. Bieda, B. Ludaescher, and B. Lud, "YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts," *International Journal of Digital Curation*, vol. 10, no. 1, pp. 1–15, 2015.

[32] J. Pimentel, V. Braganholo, L. Murta, and J. Freire, "Collecting and Analyzing Provenance on Interactive Notebooks: when IPython meets noWorkflow," *Usenix.Org*, pp. 3–8, 2015.

[33] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "Prov-dm: The prov data model," Tech. Rep., 2012. [Online]. Available: http://www.w3.org/TR/prov-dm/

[34] W. Oliveira, P. Missier, K. Ocaña, D. de Oliveira, and V. Braganholo, "Analyzing provenance across heterogeneous provenance graphs," in *International Provenance and Annotation Workshop*. Springer, 2016, pp. 57–70.

[35] AT&T. (1991) Graphviz - graph visualization software. [Online]. Available: https://www.graphviz.org/