Open Science Labs | noWorkFlow - Idea 3 - Verify the reproducibility of an experiment

Candidate Info

- **Name:** Joshua Daniel Talahatu
- **GitHub:** https://github.com/JoshuaGlaZ
- **Email:** joshuaminex02@gmail.com
- **University Course:** Informatics Engineering
- **University:** University of Surabaya
- **Time Zone:** UTC+07:00

Bio:

Hi, I'm Joshua Daniel Talahatu, currently a second-year Informatics Engineering student from University of Surabaya. I'm particularly experienced in Python programming, Git for version control, and general database management. My passion for automation fuels my interest in this project. Streamlining workflows and reducing manual tasks are core benefits of automation, which directly aligns with the goals of this project. While my core experience lies in Python for automation, I'm a fast learner and highly motivated to delve into ASTs implementation and script analysis.

Project Overview

- **Project:** Open Science Labs - noWorkflow
- **Project Idea/Plan:** Verify the reproducibility of an experiment
- **Expected Time (hours):** 300

Abstract

Develop algorithms by comparing variables value to identify reproducibility using noWorkflow.

Mentors

João Felipe Pimentel, Ivan Ogasawara

Technical Details

This project focuses on enhancing the analysis of experiment reproducibility using noWorkFlow. noWorkFlow is a library that records data lineage to from the origin to current iteration of uses in experiment trials. This includes the history of data, including its origin, transformations, and manipulations throughout the experiment. The flow of script execution / steps to run and library dependencies in the scripts also be included to the information captured by noWorkFlow. These information then be called provenance data can impact the result / outcome of the experiment even if the change is minimalistic. noWorkFlow provides functionalities to display the provenance data, access data, and give visualization for better provenance analysis. But the current methods available has limited usage when it comes to comparing trials. Current implementation for comparison only indicates the difference in structure of execution flow. A newly function `now compare` & `now analyze` is able to extend the analysis of provenance data from trials without have to explicitly and iteratively run methods for each trial id. `now compare` also give additional information for differences of values.

Compare Provenance Data

`now run` is crucial command since it run a script and saving provenance information that needed for comparison. It shows variable dependencies and their values at different points during the script's execution. There also other information such as module used, file access, function calls, and other. Trial data set then stored in SQLite database. Retrieving data needed to comparison purpose can be query using the following query.

```
SELECT trial.id AS trial_id,
       object.name AS object_name,
       object_value.value AS object_value
FROM provenance
INNER JOIN trial ON provenance.trial_id = trial.id
INNER JOIN object ON provenance.object_id = object.id
INNER JOIN object_value ON provenance.object_value_id = object_value.id
WHERE trial.id = ? AND object.name IN (?, ?)
ORDER BY object.name;
```

This query retrieves data for a specific trial ID (trial.id) focusing on comparing values for particular objects (object.name). WHERE clause can be used if needed for filtering based on specific values.

From here, I'll develop a function similar to now diff, but its core functionality focused on generating list of differences (deviations) of the trials. Data types of variable play a crucial role in the comparison process. They can range from simple data types like strings and integer to more complex structures like a dictionaries / AST containing function call information and lists of library dependencies. These data types are useful later on for determining appropriate thresholds that will influence the calculation of reproducible indicators. For instance, an error might occur when passing data2.dat as an argument, whereas previous iterations with data2.txt

might be considered within the acceptable deviation threshold based on the data type and the tolerance level defined.

The implementation on calculating the reproducibility indicators mostly factors on the differences in executions, similar to libraries like pycode_similar that calculate the percentage of plagiarism of scripts based on the reference scripts. Pycode_similar used difflib to get the differences / modified scripts from referenced. But instead of giving the percentage of similarity and only 2 scripts, `now compare` can take 2+ arguments of scripts and give a reproducibility percentage.

Comparing different executions can be used using difflib. Difflib is library that provide functions to comparing sequences. It gives an output similar to Git uses, unified diff for display of changes. Differences output should highlight modifications that could influence experiment reproducibility. This involves meticulously filtering out insignificant chan ges (e.g., whitespace adjustments) and prioritizing changes that might affect variable usage, function calls, or library dependencies.

Example Output:

```
**Trial 1 (script.py):**
* Lines Modified:
   3       import matplotlib.pyplot
   4   -
* Function Call Deviations: 2 (update_data function)
* Variable Usage Changes: 1 (variable 'threshold' now used instead of
'tolerance')

**Trial 2 (script.py):**
* Lines Modified:
       -
   3   +  import numpy
* Library Dependency Addition: 'numpy'
```

Difflib primarily focuses on line-by-line comparisons, which can be ineffective for scripts with loops, conditional statements, or function calls. These script elements might change functionality without necessarily reflecting line-level differences.Therefore, exploring Abstract Syntax Trees (ASTs) is the preferred approach for script comparisons with more complexity flow.

Reproducible Indicator Algorithm

Design of an algorithm to calculate a comprehensive reproducibility indicator heavily depends on effective thresholds for acceptable deviations and identifying critical discrepancies.The choice of similarity measure depends on the data type associated with the provenance data being compared. For comparing values of the experiment, repr of values is use to compare values that input is given. Repr provides a string representation of an object, offering a way to

check if values are identical. This can be a quick and simple approach, especially for basic data types like integers or strings. For a more detailed comparison metric, different data types can be considered for additional metric. When encounter string-like variables, Jaccard Similarity is one of the metrics used to calculate the ratio of the intersection size between two sets to the union size. Whereas numeric variables commonly used Mean Absolute Error (MAE) metric to calculate the average magnitude of the differences between corresponding values in two data sets. Abstract Syntax Trees (AST) is probably the main component of the algorithm as it covers the whole script algorithm. From the recent reading, Tree Edit Distance (TED) is currently used as a metric that extends Levenshtein Distance to compare hierarchical structures like trees as it's useful for analyzing any difference code snippets within the provenance data. TED scores the dissimilarity between scripts and pinpointing potential modifications to the script's logic that might affect reproducibility. Combining the TED and individual representation information of the values in string is used to deliver the indicator whether or not experiment trial is reproducible or not.

Analyze Experiment

`now analyze` would run the algorithm.py for that has been using data that has been collected via `now compare`. After retrieve the input of comparison trials data, It will started calculating the percentage reproducibility. A list of identified differences on variables between those trials (e.g., variable name, execution step, different values in each trial) will using Reproducible algorithm previously discussed. Result of reproducibility indicator will display on the CLI along with potential non-reproducible script snippet. Identified code snippets or variable names associated with significant deviations will be displayed using the format .

References:
https://www.nnlm.gov/guides/data-glossary/data-provenance
https://github.com/gems-uff/noworkflow
https://www.usenix.org/conference/tapp15/workshop-program/presentation/pimentel
https://docs.python.org/3/library/difflib.html
https://github.com/fyrestone/pycode_similar
https://www.researchgate.net/publication/281324160_Tree_edit_distance_Robust_and_memory-efficient

Benefit to the Community

This project will significantly benefit the research community, particularly those who rely on reproducible workflows for their experiments. noWorkFlow currently functionalities can reduce time debugging, increased confidence in results, and enhanced transparency. By enhancing noWorkflow's capabilities for reproducibility analysis, this project will encourage wider adoption of the framework within the research community. This will lead to a larger pool of researchers using standardized practices for scientific workflows, further promoting reproducibility and collaboration.

Deliverables and Timeline

Before May 1

Better understanding the topic of provenance and scripts similarity algorithms.

Community Bonding Period

- Communicate with the community and get to know the project's workflow.
- Learn noWorkFlow library by reading docs and guidelines to understand its usage specifically focusing on data access mechanisms.
- Discuss with the mentor to work out more details about the project.
- Set up a development environment and prepare blog posts for weekly reports.

Phase 1

Coding Period (May 27 – July 7)	
Week 1 May 27 – June 2	● Familiarize myself with noWorkFlow ● Understand the structure of provenance data database
Week 2 June 3 – June 9	● Design logic for comparing variable values based on data types and repr of the values.
Week 3 June 10 – June 16	● Implement `now compare` to compare provenance data by display list of differences.
Week 4 June 17 – June 23	● Implement value comparison with repr ● Research on AST processing methods
Week 5 June 24 – June 30	● Implement Mean Absolute Error (MAE) for numeric variables ● Implement Jaccard Similarity for string-like variables
Week 6 July 1 – July 7	● Implement the Tree Edit Distance (TED) algorithm to compare ASTs. ● Conduct initial testing for newly developed functions.

Evaluation Period (July 8 – July 12)

Phase 2

Coding Period (July 12 - August 19)	
Week 7 July 15 – July 21	• Continue TED algorithm with the reproducibility indicator algorithm.
Week 8 July 22 – July 28	• Define threshold of data types and discrepancies. • Start integrating `now compare` of data types for algorithm.py.
Week 9 July 29 – August 4	• Implement `now analyze` to calculate the percentage of reproducibility.
Week 10 August 5 – August 11	• Testing newly functionalities to verify and validate outputs.
Week 11 August 12 – August 18	• Improve the code quality and refactor code structure. • Further testing if needed.
Final Week August 19 – August 25	• Clean up codes and documentation. • Prepare tutorial / demos for newly implemented features.
Evaluation Period (August 26 – September 2)	

Previous Contributions to the Project

I'm relatively new to open source projects currently, So I don't have any other open source contributions.

Why this project?

The reason I’m interested in the project is in its application of script analysis techniques, particularly the ability to identify changes and highlight specific sections. The ability to analyze scripts for changes and highlight specific sections intrigue me to apply with automation. For instance, analyzing shell scripts used in automation workflows could be incredibly valuable. Furthermore, the potential applications of ASTs extend beyond automation to the development of more powerful Quality-of-Life (QoL) programs and potentially even testing frameworks.

While I'm relatively new to these specific concepts, my experience with Python programming and strong foundation in automation principles allow me to grasp them quickly. My passion for learning and eagerness to contribute to open-source development make me a fast learner who can quickly become a valuable asset to the project.

Availability

I have a student excursion to Bali starting from July 15th - July 18th, so I may have limited availability at that time. I plan to manage this by catching up on work upon return. For any other academic commitments that may arise, such as finals, I can adjust my schedule accordingly to meet project deadlines. Other than that, I have no other commitments. I plan to allocate at least 30 hours per week, during the 3 month long period.

Post GSoC

After GSoC, I'd love to remain connected with the organization and contribute to the ongoing development of noWorkFlow or other projects in OpenSouceLabs . I wanted to expand my contributions to include areas like documentation improvement or testing framework development. I believe my knowledge gained during the program, combined with my enthusiasm for automation, will allow me to provide even greater value to the project and the open-source community. To further prepare for ongoing involvement, I plan to continue exploring documentation best practices or research specific testing frameworks relevant to noWorkflow.