On Irregularity Localization for Scientific Data Analysis Workflows

Anh Duc Vu[1[0000−0003−4035−2804]], Christos Tsigkanos[2,3[0000−0002−9493−3404]], Jorge-Arnulfo Quiané-Ruiz[4[0000−0002−9001−825X]], Volker Markl[5,6[0009−0009−0964−026X]], and Timo Kehrer[2[0000−0002−2582−5557]]

[1]Humboldt-Universität zu Berlin, Germany
[2]University of Bern, Switzerland
[3]University of Athens, Greece
[4]IT University of Copenhagen, Denmark
[5]Technical University of Berlin, Germany
[6]DFKI Berlin, Germany

Abstract. The paradigm shift towards data-driven science is massively transforming the scientific process. Scientists use exploratory data analysis to arrive at new insights. This requires them to specify complex data analysis workflows, which consist of compositions of data analysis functions. Said functions encapsulate information extraction, integration, and model building through operations specified in linear algebra, relational algebra, and iterative control flow among these. A key challenge in these complex workflows is to understand and act upon irregularities in these workflows, such as outliers in aggregations. Regardless whether irregularities stem from errors or point to new insights, they must be localized and rationalized, in order to ensure the correctness and overall trustworthiness of the workflow. We propose to automatically reduce a workflow's input data while still observing some outcome of interest, thereby computing a minimal reproducible example to support workflow debugging. In essence, we reduce the problem to the determination of the input relevant to reproducing the irregularity. To that end, we present a portfolio of different strategies being tailored to data analysis workflows that operate on tabular data. We investigate their feasibility in terms of input reduction, and compare their effectiveness and efficiency within three characteristic cases.

1 Introduction

Computationally-intensive research methods exploiting large amounts of data are becoming ubiquitous in numerous scientific disciplines [12]. During the scientific process, researchers leverage exploratory data analysis to analyze, manipulate, and investigate data sets in order to apply statistical techniques, spot anomalies, test hypotheses, or check assumptions. Typically, this involves the composition of heterogeneous collections of data processing functions (e.g., data integration, normalization, and filtering) into complex data analysis workflows [16]. Exploratory analysis in scientific computing often yields results for which it is

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

hard to judge whether they are correct or not [23]. Irregularities in workflow results could point to new interesting insights, or may be caused by errors in the workflow or corrupt input data [5]. Either way, one has to perform data debugging [3] with the goal of finding the cause of irregularities in order to increase the trustworthiness in the workflow [21].

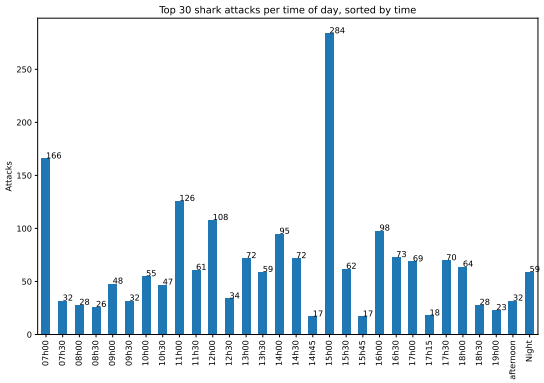
Many researchers pursued debugging and validation in data processing systems from different angles, notably by allowing the inspection of intermediate [8] or sub-results [6], or tracking data provenance [2,13,14] for dataflow systems like Apache Spark or Flink. What is common among these data provenance and debugging approaches is that they assume that the workflow is a white-box: they must be able to follow and label tuples through the entire process. However, this is far from reality in scientific data analytics, where workflows are mostly composed of black-box processes. Moreover, classical fault localization techniques, e.g., [1,9,26], assume that suspicious behavior manifests in an error that can be identified by a pre-defined test. Nevertheless, scientific data analytics is explorative by nature, making it often impossible to define tests that specify correct behavior in terms of expected outcomes [15,19].

Conversely, we aim at finding the cause of an irregularity in a workflow's input data by computing a *minimal reproducible example* that produces the irregularity. The general idea is to iteratively reduce the input data to ease the task of finding the cause of the output's irregularity. As opposed to classical fault localization, we refer to this process as *irregularity localization*, emphasizing the fact that the outcome investigated may or may not be faulty.

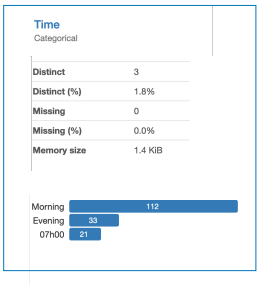
Recently, we proposed *outcome-preserving input reduction* as a generic approach to support irregularity localization in data analysis workflows [24]. The idea is to iteratively reduce the workflow's input data while still observing some outcome of interest in its results. While we do not ask developers for pre-defined tests, we assume that, by looking at the workflow's result, they may specify a debugging question that characterizes the outcome of interest. For instance, the expressions "$(result.Time = \text{"07}h\text{00"} \wedge result.Count \geq 166)$" may be questioning a counted number in an aggregation. Then, the workflow may be executed on a reduced input, and the debugging question is evaluated on its output data in order to check whether a reduction is outcome-preserving and thus permitted. For example, whether "$(result.Time = \text{"07}h\text{00"} \wedge result.Count \geq 166)$" still holds in the new result. We intend to perform this in iterations until we reach a certain fixpoint (e.g., a degree of reduction or the consumption of a given resources budget), thus obtaining a minimal reproducible example as a result.

While the framework advocated in [24] is generally applicable, as it abstracts from workflow implementations and execution engines, it needs to be instantiated by providing implementations of its components. In particular, implementing a reducer facility is challenging. The large amount of data and the limited information about the workflow's underlying mechanism make it difficult to distinguish the relevant parts of the input data and require careful selection strategies for the reduction attempts. In this paper, we present a portfolio of reduction strategies tailored to scientific data analysis workflows, making the following contributions:

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24



(a) Result with a suspiciously high number of attacks at 07h00.



(b) Data profile of a minimal reproducible example.

Fig. 1: An example workflow that aggregates shark attacks per time of day.

- We present a concrete instantiation of a general framework for irregularity localization in data analysis workflows (Sec. 3);
- We introduce a portfolio of strategies targeting the reduction of tabular data. In particular, we propose Similarity-based Isolation, a data reduction strategy that isolates those tuples, together with all their similar tuples, having an effect on the outcome of interest (Sec. 4).
- We evaluate the proposed strategies experimentally over three different cases with respect to feasibility and their individual performance (Sec. 5);
- We provide a replication package allowing our results to be reproduced.

2 Motivation and Background

Consider a data analysis workflow implemented as a computational notebook taken from Kaggle[1] over a dataset comprising shark attacks worldwide in the last 100 years, where the workflow seeks to figure out the number of attacks per time of the day. Fig. 1a shows a plot of the workflow's result. Notably, for 07h00 the number of attacks appears to be quite high compared to other times of frequent attacks, which are around noon and afternoon – although the peak at 15h00 is also high, the attacks at 07h00 are at a suspicious time. It may certainly be the case that the data indeed show a high amount of attacks for that time; however, there may also be an error or wrong assumption in the analysis performed to produce this result, or the data may be corrupt. Either way, there is an aspect of the result which is suspicious, i.e., showing an *irregularity* which we should investigate further. In [24] we argued that such investigation of suspicious results within data analysis workflows must accommodate the specific needs of the scientists developing these workflows. Like the scientific discovery process itself, the investigation should be done in an explorative manner. Our

[1] kaggle.com/mysarahmadbhat/shark-attacks

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

example represents a characteristic case where exploratory analysis is required to deduce if the data indeed support shark attacks occurring early in the morning. Intuitively, one would seek to spot the cause of the suspicious outcome with the help of a minimal example of the input, referred to as *irregularity localization*.

The dataset of our motivating example contains about 25k rows and 24 columns, rather small compared to what is typically processed within scientific data analytics, but still overwhelming for a human being. Thus, it is desirable to reduce the input dataset to the minimum set of rows and columns of the input table that reproduce the outcome of interest. We refer to this task as *outcome-preserving input reduction*. For our example case, it turns out that we require only a minor fraction of the input dataset to enable the workflow to reproduce the suspicious result. Fig. 1b depicts a profile of such a reduced input dataset, comprising only 166 tuples and one column that are needed to reproduce the peak of irregularly many shark attacks at 07h00. Observe that only column 'Time' has been deemed relevant, and the relevant tuples take the values of 'Morning', 'Evening' and '07h00', which is surprising and would have been difficult to find manually. This *minimal reproducible example* provides a strong hint to the user; some data transformation applied within the workflow did yield these irregular values, which needs to be investigated. For this particular example workflow, the seemingly high number of attacks is caused by a data cleaning function that attempts to map textual reports to clock time. Apparently, the workflow interprets the string values "Morning" and "Evening" to 7 o'clock (which is a questionable assumption made by the workflow developer), and it misses to distinguish a.m. from p.m. (which is clearly an error in the workflow's implementation).

Without this automated reduction, users may try to query the input data by selecting the tuples where 'Time = 07h00' to investigate their suspicion on the workflow's output. The result, however, would be that only 21 tuples of the input that actually have the value '07h00' would show up – as seen in Fig. 1b – giving the user no indication where the remaining 145 attacks come from. More generally, the key challenge addressed by our approach is to support irregularity localization in cases where the suspicious parts of the output are a result of the workflow's execution, but they cannot be easily spotted in the input data, rendering any ad-hoc analysis of the input data infeasible – especially when the workflow contains tasks that change the data before processing it.

2.1 General Framework for Outcome-Preserving Input Reduction

A high-level vision of how to systematically support irregularity localization through outcome-preserving input reduction for data analysis workflows is illustrated in Fig. 2. A scientist initiates a workflow by submitting some input data, yielding an output (marked as (1)). On further inspection by the scientist, certain parts or aspects of the output data may be of particular interest, perhaps because they are perceived as suspicious or exhibit irregularities that the scientist did not expect and lacks a reasonable explanation for (2).

A debugging question – in essence a query specified by the user that formally describes the outcome of interest – together with a termination condition – e.g.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

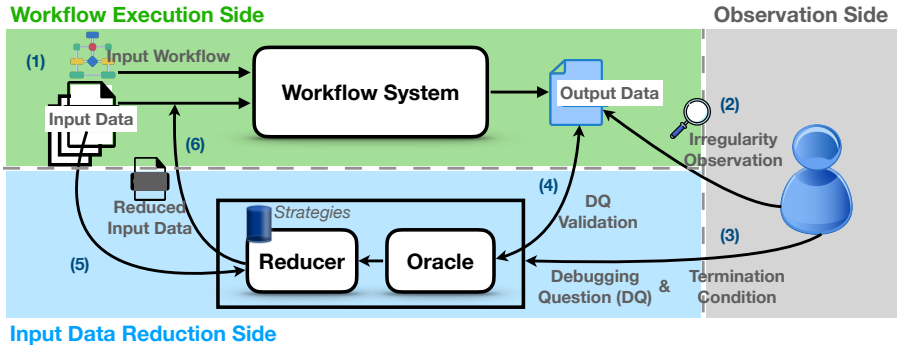


Fig. 2: Outcome-preserving input data reduction for irregularity localization.

minimality of the input – is then supplied to the input reduction facility (3). The debugging question is used by the `Oracle` component to generate an assertion to answer whether the outcome specified by the question holds on the output data (4). The difference to classical test oracles in software testing is that, in general, there is no assessment on whether the specified outcome is correct or not. Moreover, whereas tests are specified beforehand to define expected outcomes, debugging questions are meant to be used ad-hoc on any observed outcome, making them tools for explorative investigation.

The `Reducer` component operates upon the input data facilitating its reduction strategies in order to yield a subset of it (5). The analysis workflow may be triggered again with the reduced input, producing a new output (6). Thereupon, again the oracle is employed to decide if the outcome of interest is still observable in the new output data. This process is repeated iteratively until the termination condition holds (4)-(6). The result (at each iteration) is a reduced input intended to aid understanding the circumstances involving the irregularity that is sought to be investigated.

3 Instantiation of the General Framework

To serve as a practical solution, the general framework previously introduced needs to be instantiated by providing concrete implementations of (i) the formalism for specifying debugging questions to be interpreted by an oracle, and (ii) the strategy of the reducer. Both heavily depend on the structure of the workflow's input and output. In this paper, we focus on tabular data, widely used in data science and scientific computing. We assume that a workflow takes a single dataset as input and produces a single output dataset. Both input and output data are in the shape of a table, comprising arbitrary rows over a fixed number of columns whose data types are defined by a schema.

Debugging Question and Oracle. When a user wishes to investigate their suspicion regarding a result, the manner on how they express and formalize the observed irregularity poses a challenge. Assuming the output is in the form of a table, we propose to use an "SQL-like" query to characterize the suspicious parts

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

of the table. With ease of use and familiarity in mind, we select the query syntax of the python software library pandas[2]. This allows for a smoother adoption of the technique advocated, as it eliminates the need for most users to learn yet another new syntax. In our motivating example, for the unusual high number of shark attacks at 7 o'clock in the workflow's result, we may formulate the pandas query '(Time=="07h00" & Count == 166)'. An oracle evaluates the debugging question on the output table, obtaining a certain number k of output tuples. We store this amount of output tuples when the debugging question is evaluated on the output table obtained from running the workflow on the original input data set. Later on, after every iteration, the same query is then evaluated on the output table obtained from running the workflow on a reduced input data set. The outcome of interest is preserved if the query yields the same amount of k output tuples.

Outcome-Preserving Input Data Reduction. Reducing an input dataset to a minimal one where any smaller subset does not yield the outcome of interest anymore presents a conceptual challenge. Without any assumptions on the workflow behavior, finding such a minimal input dataset requires an exhaustive search, which is infeasible for real-world datasets comprising a large number of tuples. We can ease the problem by only requiring a local minimum which, however, still requires testing its exponentially many subsets. A similar problem that suffers from such a combinatorial explosion has been described by Zeller et al. [26] in the context of test minimization. To keep the search for a local minimum tractable, even when the search space is large, an approximation based on the notion of *n-minimality* is defined which we adapt for our setting as follows:

Definition 1 (n-minimality). *Let I be a set of tuples, A be a subset of I, and $is\_preserving : 2^I \rightarrow \{True, False\}$ be an evaluation function with $is\_preserving(A) = True$. A is* n-minimal *if for all $B \subset A$ it holds that if $|B| \leq n$ then $is\_preserving(A - B) = False$.*

In the use case of test minimization, [26] argues for *1-minimality* being sufficient to aid the developer to investigate what exactly causes the test case to fail. We adopt a similar viewpoint for irregularity localization. If the user is presented with a reduced input dataset where each tuple is necessary to reproduce results obeying the outcome of interest, they can either: (i) surmise that the parts of the workflow processing the data in question are not behaving as intended, or (ii) improve their understanding of the workflow for further validation.

4 Investigated Reduction Strategies

We now describe the algorithms employed to find *1-minimal* outcome preserving sets. We first review some baseline strategies: `leave-one-out`, `dd-min`, `prob-dd`; and afterwards describe a novel strategy based on similarity search and fault isolation (`similarity-iso`).

[2] https://pandas.pydata.org/

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

4.1 Baseline (leave-one-out)

To find a 1-minimal set, a naive strategy consists of iteratively removing a single data element from the input data set, running the workflow, and consulting the oracle of whether the outcome of interest is preserved. This step is applied to all the elements from the input data set until an element removal changes the outcome, and the next iteration starts until no more elements can be removed. In our approach, it serves as the baseline upon which the strategies outlined in the following are to be compared.

4.2 Delta Debugging (dd-min)

Our second strategy is based on the principle of *Delta Debugging*. Specifically, we adapt `ddmin` [26], a classical fault localization technique originally proposed for test case minimization. `ddmin` is based on a binary search: we start by splitting the input space into two halves and testing each of them whether they can be removed from the input without changing the outcome. If one of the halves can be removed, we found a new smaller dataset that still preserves the outcome. We continue halving the smaller but still outcome preserving input partition. If neither of the halves preserve the outcome, the granularity is increased to remove quarters of the dataset. This is done until the granularity is so small that we try to remove single elements from the dataset, at which point it becomes the leave-one-out strategy.

4.3 Probabilistic Delta Debugging (prob-dd)

Wang et. al. [25] pointed to the weakness of the `ddmin` algorithm in not taking advantage of past executions while iterating, but only following a pre-determined schedule of testing subsets. To ameliorate this, they devised *Probabilistic Delta Debugging*; the improvement consists of using a probabilistic model to guide the selection which is updated after each test. In such a model, each element of the input is assigned a random variable that decides whether that element will be in the reduced dataset or not. We refer to the original paper [25] for technical details.

4.4 Similarity-based Isolation (similarity-iso)

The next strategy is constructed with the objective of better localizing elements of interest than `ddmin`. It is based on the intuition that similar elements – by means of a distance function – will cause similar effects on the outcomes of a computation. This strategy is a derivative of the *dd* fault isolation algorithm [26], which computes an *n-minimal-difference* between two sets.

Definition 2 (n-minimal-difference). *Let I be a set of data tuples; A, B are subsets of I, and $is\_preserving : 2^I \rightarrow \{True, False\}$ is an evaluation function. Given that $B \subset A$, $is\_preserving(A) = True$, and $is\_preserving(B) = False$, then $A - B$ is an* n-minimal-difference *if for all $C \subseteq (A - B)$ it holds that if $|C| \leq n$ then $is\_preserving(A - C) = False$ and $is\_preserving(B \cup C) = True$.*

Our strategy exploiting the notions of a 1-minimal difference and similarity of data tuples is illustrated in Algorithm 1. The outer while loop (lines 3 to 16) maintains two sets. `min_preserving` refers to the smallest input dataset

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

Algorithm 1: Similarity-based Isolation

```
   Data: input
   Result: reduced_input
 1 min_preserving ← input
 2 isolated ← ∅
 3 while True do
       /* Isolate a 1-minimal difference */
 4     max_changing ← ∅
 5     Δ ← (min_preserving − max_changing) − isolated
 6     while |Δ| > 1 do
 7         test ← (get_subset(Δ, ½|Δ|) + max_changing) + isolated
 8         if is_preserving(test) = True then
 9             min_preserving ← test
10         else
11             max_changing ← test
12         Δ ← (min_preserving − max_changing) − isolated
13     if Δ = ∅ then
           /* No new tuples can be isolated */
14         break
15     else
           /* Isolate Δ and all similar tuples in min_preserving */
16         isolated ← isolated + Δ + get_similar_elements(min_preserving, Δ)
17 return ddmin(isolated)
```

that preserves the outcome of interest; it is initialized with the original input dataset (line 1) and constantly reduced within each iteration. `isolated` is initialized with the empty set (line 2) and accumulates those input tuples which, presumably, have an effect on the outcome. In each iteration, `min_preserving` and `isolated` are updated in two steps: First, we try to find a single tuple that is not yet included in `isolated` (lines 4 - 12) but causes a different outcome if removed. Second, if such a tuple can be found, this tuple as well as all similar – by some distance function, e.g. levenshtein – tuples in `min_preserving` that presumably also change the outcome are added to `isolated` (line 16). This process of alternating isolation and similarity search is repeated until no new elements can be isolated (line 14) and a fixpoint is reached. The procedure finishes by running any of the other minimization strategies – currently `ddmin` – to find a *1-minimal* solution (line 17).

Isolation: The isolation step (lines 4 - 12) is realized by isolating a 1-minimal difference, referred to as Δ, between `min_preserving` and `max_changing`, a set that accumulates the currently largest set of input tuples changing the outcome of interest. Since we can safely assume that the empty set produces a radically different result than the original input data set and thus changes the outcome of interest, `max_changing` is initialized with the empty set (line 4). The set Δ is initialized as the set difference between `min_preserving` and `max_changing`, from which we also remove those tuples that have been isolated in previous iterations (line 5). Then, in each iteration of the inner while loop (lines 6 - 12), half of

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

the elements are taken from Δ, and unified with the currently largest outcome-changing set `max_changing` and the already isolated elements in `isolated` (line 7). For the obtained set of input tuples, referred to as `test`, we run the workflow and let our oracle decide whether the outcome of interest is preserved (line 8). If so, a smaller outcome-preserving set has been located (line 9). Otherwise, a new largest outcome changing set has been located (line 11). We proceed with the next iteration of the inner while loop, as long as Δ contains more than one tuple (line 6). The procedure takes logarithmic many steps to isolate an *1-difference*.

Similarity Search: The similarity search (line 16) follows the intuition that similar data is likely to have similar effects on the outcome of a computation. Given our assumption of tabular data, we define the similarity of two data records as the average similarity over each column value For each column value, we employ classical distance metrics, depending on the column's data type. In the base case, we currently use normalized Euclid distance for numerical values, Levenshtein distance for strings, and equality for categorical data.

5 Evaluation

We conducted experiments to assess how suitable it is to perform outcome-preserving input reduction with our framework and the aforementioned strategies. Our evaluation[3] addresses the following research questions: (**RQ1**) How feasible is a search for 1-minimal datasets for outcome-preserving input reduction?, and; (**RQ2**) How do the reduction strategies compare to each other with respect to their resource consumption? To answer RQ1, we consider measures from the user's perspective – namely the degree of reduction and the overall time it takes to do so. For RQ2, we investigate the runtime properties of the reductions, measuring the number of times the workflow is executed by a strategy, and the time a strategy requires between each of these iterations.

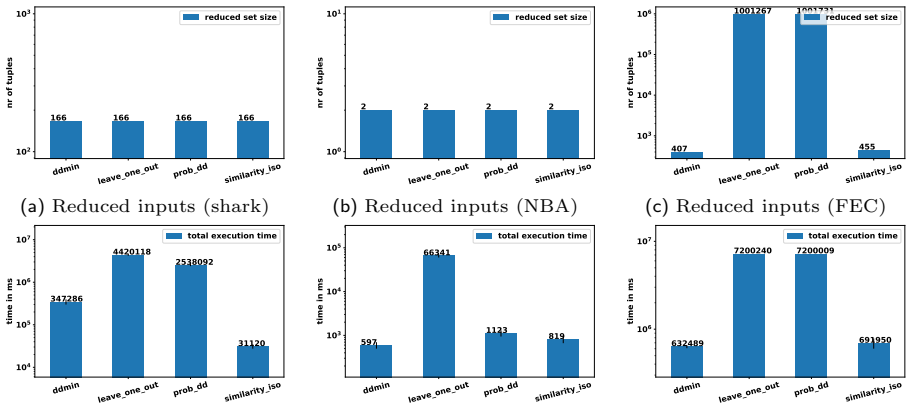
5.1 Experimental Setup

We adopt three workflows derived from publicly available computational notebooks, referred to as Shark (described in Sec. 2), NBA (a workflow analyzing NBA players), and FEC (a workflow analyzing the 2012 US federal election commissions data).

The NBA case is an erroneous notebook from a study about failure identification strategies in Jupyter notebooks [20], which entails explorative analysis on a dataset about NBA players – study participants were tasked with fixing various errors. At some point the NBA players are grouped into point and shooting guards, and their average height must be calculated. The study mentions that one of the participants remarked that a "mean height of 12 doesn't seem to make a lot of sense", likely referring to an expected length unit of centimeters or meters. As such, we adopt the parts of the NBA notebook calculating the average height, and apply our search strategies with the debugging query (Position ==

[3] Our replication package can be found at `https://osf.io/fk2x4/?view_only=442434edaec94c2b8172a759699d0886`

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24



(a) Reduced inputs (shark) (b) Reduced inputs (NBA) (c) Reduced inputs (FEC)

(d) Exec. times (shark) (e) Exec. times (NBA) (f) Exec. times (FEC)

Fig. 3: Size of reduced input in no. of tuples and total execution time in ms (log scale).

"PG" $\wedge$ Height < 100) to check why point guards appear with such low height. The dimensions of this dataset are 1184 rows, and 13 columns.

The third case concerns the US federal election commissions data from 2012 and is taken from the book "Python for Data Analysis, 3rd Edition"[4]. The input dataset consists of about 1M rows and 16 columns. One of the results of this analysis yields a table listing the donations by state to candidates Barack Obama and Mitt Romney. Of interest is that for the state Arkansas, Obama received about 77% of the donations from individuals, more than three times as much as Romney. However, in the election of 2012, Arkansas was won by the Republican party with more than 60% of the votes – a possible irregularity. Therefore, we use the following debugging question to investigate: '$contbr\_st$' == "AR" $\wedge$ '$Obama, Barack$' $> 0.77 \wedge$ '$Obama, Barack$' < 0.78.

We proceed to apply the reduction strategies of Sec. 4. Experiments were performed on an Intel E7-4880 2.5GHz CPU. Each experiment was repeated 10 times; measurements reported in the sequel are average values.

5.2 Results

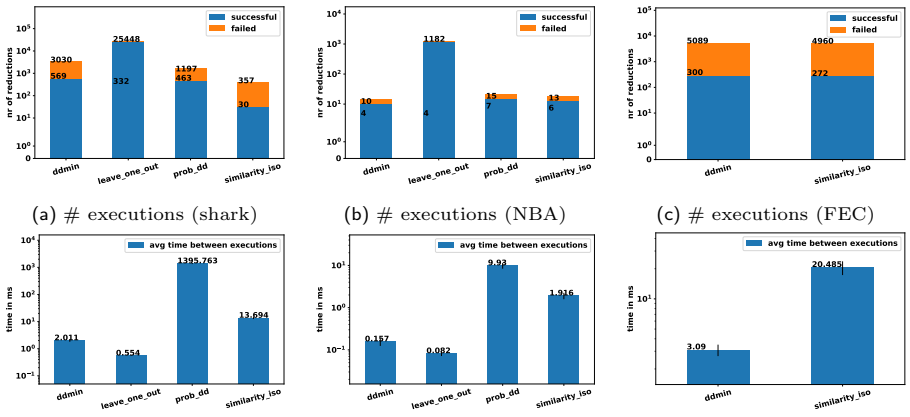
The issue that causes the Shark workflow to calculate a surprisingly high number of attacks has been already discussed in Sec. 2. A minimal example to reproduce this result is a dataset that comprises the 166 tuples taking the values "7h00", "Morning", and "Evening" in the column 'Time'.

The low average height calculated by the NBA workflow is caused by a data cleaning step which neglects that players' heights are given not only in centimeters but also in inches. So any input set of size 2 where at least one of the elements has a height value in inches is sufficient to reproduce the outcome specified.

The FEC workflow was obtained with no fault injection – a reduced input set should confirm that the analysis is as intended, illustrating that the presented

[4] wesmckinney.com/book/data-analysis-examples.html

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24



(a) # executions (shark) (b) # executions (NBA) (c) # executions (FEC)

(d) Avg. time between executions (shark) (e) Avg. time between executions (NBA) (f) Avg. time between executions (FEC)

Fig. 4: Number of workflow executions and average time between executions (log scale).

approach is not only useful for irregularity localization but also for validation by providing a smaller dataset.

RQ1. Sizes of the reduced datasets and the time needed to conclude the reductions are reported in Fig. 3. Note that the y-axes are scaled logarithmically. For the Shark and NBA workflow, each strategy found 1-minimal input sets of the same size, namely 166 and 2 tuples, which corresponds to reductions of 99,5% and 99.8% compared to the original input sizes of 25614 and 1184, respectively. Regarding the overall time needed to find these 1-minimal sets, `leave-one-out` took the longest, with a factor of almost 100 compared to the other startegies in case of the NBA workflow. Also `Prob-dd` was clearly outperformed by `ddmin` and `similarity-iso` on both cases. For the shark workflow, `prob-dd` took around 40 minutes, `ddmin` required about 6 minutes, and `similarity-iso` was the clear winner requiring less than a minute. For the NBA case, `ddmin` requires around 600ms, which is about 25% less than the second best strategy `similarity-iso`. In the FEC case, we observe that `ddmin` with a runtime of 8 minutes is about 1 min faster than `similarity-iso` and produces a dataset about 10% smaller. `leave_one_out` and `prob-dd` are not competitive at all within a time-out of 2 hours. Because the number of tuples in the FEC dataset is so high, `leave_one_out` is getting timed out because it is only removing a single tuple per iteration, and `prob-dd` takes so long to update probabilities for each element that it also times out before it can finish. In the sequel, we will thus not consider these strategies for the FEC workflow evaluation, and they are omitted from Fig 4f and Fig. 4c, respectively.

RQ2. As observed, there are differences in how long the various strategies require to find a solution. Fig. 4 illustrates RQ2-related measurements, again noting that the y-axes are log-scaled. Figure 4a shows the number of reductions that were attempted by each strategy for the shark workflow, divided by successful and failed attempts – the sum represents the total number of workflow executions. Unsurprisingly, the longest running strategy `leave-one-out` also executed

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

the workflow the most, while the fastest strategy `similarity_iso` required the least amount of attempts. Interestingly, even though `prob-dd` only took about half as many reduction attempts than `ddmin`, it took nearly 7 times as long. For an explanation, observe Fig. 4d, depicting the average execution time between workflow executions. This metric records the average time each strategy requires to decide on the next reduction candidate based on the result of the previous execution. For this metric, we observe that `prob_dd` requires considerably more time to choose the next candidate, but with the payoff of needing only half as many executions than `ddmin`. The same can be observed with `similarity-iso`, which takes more than 6 times as much to choose the next potential solution than `ddmin`, but requiring 10 times less workflow executions. For the NBA case, Fig. 4e illustrates that the longest running strategy also executed the workflow the most, with `leave_one_out` doing so almost more than 100 times as much as the others. Again, `prob-dd` takes considerably more time between workflow executions. `ddmin` requires the least amount of executions. Paired with a low overhead between the executions, this allows `ddmin` to be faster. The FEC workflow shows that even though `similarity-iso` did less iterations than `ddmin`, the higher time between iterations cause `similarity-iso` to perform a bit slower.

5.3 Summary and Threats to Validity

Our results, when viewed *ex post facto*, illustrate some key findings. Firstly the baseline strategy `leave-one-out` performs the worst and is impractical, especially for a dataset like FEC that has millions of tuples – as expected for a strategy that removes only a single element each iteration. Secondly, when relevant data (and resp., irrelevant data) is spatially grouped together, a simple partitioning strategy like `ddmin` can easily remove unnecessary elements,. However, in the shark example, where relevant tuples are scattered, `ddmin` underperforms. Instead, our novel strategy `similarity-iso` is able to isolate the scattered data quickly, vastly outperforming `ddmin` in those cases, while still being competitive with `ddmin`'s best cases by only adding minimal overhead. `prob-dd` is performant at choosing promising candidates while simultaneously being generically applicable. However, the amount of time and space needed to update element probabilities can be costly when the number of elements is large, which we observed in the FEC workflow, where the high number of tuples caused `prob-dd` to timeout. Overall, our results show that similarity search is rather balanced and performant, concluding up to 10 times faster in cases that pose difficulties for the others, while where `ddmin` outperforms, `similarity-iso` is only marginally slower while still requiring less iterations. The above strengthen the argument that `similarity-iso` offers the best tradeoff and has the strongest potential for further big data refinements.

Regarding threats to validity, risk of internal validity mainly lies in the correctness of implementation (which we release along with a reproduction kit). The core implementations of `ddmin` and `prob-dd` are based on their respective original sources with instrumentation for measurements and time-outs. Risk to construct validity lies in the fact that the debugging questions, which are central to the approach, have been specified by us. But, since this technique is mainly

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

supposed to be a tool for the exploration of the workflow behaviour, any debugging question that specifies a property of the result could have been taken to showcase the reduction strategies. As such, finding the "correct" debugging question, is out of the scope of this work. Another possible point is the number of experiment repetitions; three of the four strategies however are deterministic, and `prob-dd`'s main hurdle is the amount of processing between iterations. The threat to external validity lies in the subject selection – vastly different cases, e.g., where tuple similarity is difficult to be defined, may yield different results. However, it may be argued that the cases presented are representative of typical workflows utilizing tabular data.

6 Related Work

Much of the research on scientific workflows has focused on optimizing for speed and resource utilization, led by high-performance computing [17, 22]. However, with workflows becoming ubiquitous, there is a changing mindset that human productivity arguably still is the most expensive resource [4].

Artemis [11] and Nautilus [10] are systems that provide explanations of why or why not certain tuples are in the result, by modifying sub-query operators or by inserting tuples into the data. In our black box setting we cannot modify the individual operators of a workflow and we do not address why a result is missing but rather why a particular result is there using only the input data.

So-called why-provenance [2] is popular to answer why a specific tuple appears in the result, representing the set of input tuples that are responsible for its computation. Ikeda et al. [13] demonstrate how to utilize provenance to debug workflows by enabling forward tracing of input tuples and backward tracing of result tuples. Titian [14] enables collection of provenance for Apache Spark by tracking records through the various data operators. TagSniff [3] is a data debugging model for big data that allows users to efficiently capture data provenance. However, given the black-box nature of scientific workflow tasks, the above works are not applicable because they require white-box access to either track or reverse calculate the relevant tuples. Gulzar et. al. [7] also remark that provenance often returns excessively much data, and in the worst case the whole input. To tackle this, their BigSift approach combines provenance with delta debugging to find a minimal set that leads to a test failure, which relies on a test suite that cannot be generally assumed to be available.

BugDoc [18] can find the responsible component in a pipeline when given an oracle, by iteratively re-executing it with different instantiations. We note that our technique can be used after BugDoc identifies the responsible input data, to reduce it to a minimal reproducible example for further cause localization.

7 Conclusion and Future Work

Motivated by the need to support exploratory analysis within scientific computing, we proposed to iteratively reduce a workflow's input data while still

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

observing some outcome of interest to produce a minimal reproducible example – in essence, we determine the input relevant to reproducing the irregularity. To that end, we presented a portfolio of reduction strategies applicable to tabular data, the shape most often encountered by scientist users. We further investigated the strategies' input reduction and resource consumption over three case studies. To realize an end-to-end framework, we identify open challenges at different levels of abstraction, which provide avenues for future work. Firstly, we considered tabular data; in general, data shape as well as processing and size naturally affect choice and development of reduction strategies, yielding tradeoffs that should be assessed. Thus, future work should investigate the applicability on larger sizes of data, and also data across different domains like imaging and genomic data for example. Secondly, regarding the debugging questions, it is true that the burden of formulating them still lies on the user. We argue that our use case is slightly easier than the well known oracle problem from software testing, where a user has to specify correct behaviour upfront. In the presented use case, the user observes already computed results from the workflow and may formulate properties that this data has right now, as opposed to properties that all results should have. Thus, instead of thinking about all possible results, the user only has to consider the currently observed data. Still, the important step of the debugging question specification should be investigated; users should be supported effectively in their formulation in subsequent works.

Acknowledgements. Funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1404 FONDA

References

1. Abreu, R., Zoeteweij, P., Van Gemund, A.J.: On the accuracy of spectrum-based fault localization. In: Testing: Academic and industrial conference practice and research techniques. IEEE (2007)
2. Buneman, P., Khanna, S., Wang-Chiew, T.: Why and where: A characterization of data provenance. In: Intl. Conf. on Database Theory. Springer (2001)
3. Contreras-Rojas, B., Quiané-Ruiz, J., Kaoudi, Z., Thirumuruganathan, S.: Tagsniff: Simplified big data debugging for dataflow jobs. In: ACM Symposium on Cloud Computing. pp. 453–464. ACM (2019)
4. Deelman, E., et al.: The future of scientific workflows. Journal of High Performance Computing Applications **32**(1) (2018)
5. Galhotra, S., Fariha, A., Lourenço, R., Freire, J., Meliou, A., Srivastava, D.: Dataexposer: Exposing disconnect between data and systems. arXiv preprint arXiv:2105.06058 (2021)
6. Grust, T., Kliebhan, F., Rittinger, J., Schreiber, T.: True language-level SQL debugging. In: Intl. Conf. on Extending Database Technology (2011)
7. Gulzar, M.A., Interlandi, M., Han, X., Li, M., Condie, T., Kim, M.: Automated debugging in data-intensive scalable computing. In: Symposium on Cloud Computing (2017)
8. Gulzar, M.A., Interlandi, M., Yoo, S., Tetali, S.D., Condie, T., Millstein, T., Kim, M.: Bigdebug: Debugging primitives for interactive big data processing in spark. In: ICSE. IEEE (2016)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24

9. Heiden, S., Grunske, L., Kehrer, T., Keller, F., Van Hoorn, A., Filieri, A., Lo, D.: An evaluation of pure spectrum-based fault localization techniques for large-scale software systems. Software: Practice and Experience **49**(8), 1197–1224 (2019)
10. Herschel, M., Eichelberger, H.: The nautilus analyzer: understanding and debugging data transformations. In: Intl. Conf. on Information and Knowledge Management. pp. 2731–2733 (2012)
11. Herschel, M., Hernández, M.A.: Explaining missing answers to spjua queries. Proceedings of the VLDB Endowment **3**(1-2), 185–196 (2010)
12. Hey, A.J., Tansley, S., et al.: The fourth paradigm: data-intensive scientific discovery, vol. 1. Microsoft Research (2009)
13. Ikeda, R., Cho, J., Fang, C., Salihoglu, S., Torikai, S., Widom, J.: Provenance-based debugging and drill-down in data-oriented workflows. In: Intl. Conf. on Data Engineering. IEEE (2012)
14. Interlandi, M., Shah, K., Tetali, S.D., Gulzar, M.A., Yoo, S., Kim, M., Millstein, T., Condie, T.: Titian: Data provenance support in spark. In: Proc. of VLDB. vol. 9 (2015)
15. Kanewala, U., Bieman, J.M.: Testing scientific software: A systematic literature review. Information and software technology **56**(10) (2014)
16. Leser, U., Hilbrich, M., Draxl, C., Eisert, P., Grunske, L., Hostert, P., Kainmüller, D., Kao, O., Kehr, B., Kehrer, T., Koch, C., Markl, V., Meyerhenke, H., Rabl, T., Reinefeld, A., Reinert, K., Ritter, K., Scheuermann, B., Schintke, F., Schweikardt, N., Weidlich, M.: The Collaborative Research Center FONDA. Datenbank-Spektrum (1610-1995) (November 2021)
17. Lin, B., et al.: A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. IEEE Trans. on Industrial Informatics **15**(7) (2019)
18. Lourenço, R., Freire, J., Shasha, D.: Bugdoc: A system for debugging computational pipelines. In: Proc. of the 2020 ACM SIGMOD (2020)
19. Pimentel, J.F., Murta, L., Braganholo, V., Freire, J.: A large-scale study about quality and reproducibility of jupyter notebooks. In: Intl. Conf. on Mining Software Repositories. IEEE (2019)
20. Robinson, D., Ernst, N.A., Vargas, E.L., Storey, M.A.D.: Error identification strategies for python jupyter notebooks. arXiv preprint arXiv:2203.16653 (2022)
21. Sanders, R., Kelly, D.: Dealing with risk in scientific software development. IEEE software **25**(4) (2008)
22. Shirvani, M.: A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems. Engineering Applications of Artificial Intelligence **90** (2020)
23. Vogel, T., Druskat, S., Scheidgen, M., Draxl, C., Grunske, L.: Challenges for verifying and validating scientific software in computational materials science. In: Intl. Workshop on SE for Science. IEEE (2019)
24. Vu, A.D., Kehrer, T., Tsigkanos, C.: Outcome-preserving input reduction for scientific data analysis workflows. In: Intl. Conf. on Automated Software Engineering, New Ideas and Emerging Results (2022)
25. Wang, G., Shen, R., Chen, J., Xiong, Y., Zhang, L.: Probabilistic delta debugging. In: ESEC/FSE (2021)
26. Zeller, A., Hildebrandt, R.: Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering **28**(2) (2002)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-35995-8_24