Supporting Undo and Redo in Scientific Data Analysis

Xiang Zhao[UM] Emery R. Boose[Harvard] Yuriy Brun[UM] Barbara Staudt Lerner[MHC] Leon J. Osterweil[UM]

[UM] *University of Massachusetts, Amherst*

[Harvard] *Harvard University*

[MHC] *Mount Holyoke College*

Abstract

This paper presents a provenance-based technique to support undoing and redoing of data analysis tasks. The technique targets scientists who experiment with combinations of approaches to processing raw data into presentable datasets. Raw data may be noisy and in need of cleaning, it may suffer from sensor drift that requires retrospective calibration and data correction, or it may need gap-filling due to sensor malfunction or environmental conditions. Different raw datasets may have different issues requiring different kinds of adjustments, and each issue may potentially be handled by different approaches. Thus, scientists must often experiment with different sequences of approaches. In our work, we show how provenance information can be used to facilitate this kind of experimentation with scientific datasets. We describe an approach that supports the ability to (1) undo a set of tasks while setting aside the artifacts and consequences of performing those tasks, (2) replace, remove, or add a data-processing technique, and (3) redo automatically those set aside tasks that are consistent with changed technique. We have implemented our technique and demonstrate its utility with a case study of a common, sensor-network, data-processing scenario showing how our approach can reduce the cost of changing intermediate data-processing techniques in a complex, data-intensive process.

1 Introduction

Environmental science has been significantly advanced by the advent of sensor networks, which make it possible to collect unprecedented amounts of information. But with this advance come new challenges related to the size of the datasets, the urgency of handling streaming data, and the complexity of near-real-time quality control. Though the raw sensor data are typically archived unchanged, data products of value to users must go through a series of complex transformations, which may be revisited at a later time as more information becomes available. This presents a problem for data providers, who, in an ideal world, would carefully document how different versions of the data were derived and when they were made available to users (who, in turn, may, for example, have used a particular version in a publication). In practice, this is rarely done because of the time and effort required. There is a pressing need for automated systems that provide this service through the use of provenance.

In this paper, we describe a simple case study that nevertheless captures some of the key data transformations required for sensor data and permits the scientist to update these transformations over time. The example includes three transformations that must be performed in the following order:

(1) **Sensor calibration**. Many electronic sensors are subject to drift over time and must be periodically re-calibrated or replaced. In some cases (e.g., where there are redundant sensors), it may be possible to check and correct sensor readings in real time. But more often, the sensor is periodically returned to the manufacturer for recalibration. The scientist may then adjust previous readings by using new information on how the sensor has drifted over time.

(2) **Quality control**. Quality control procedures for sensor data include tests for outliers (range), excessive rate of change (slope), and repeated values (constant). For example, the climatic history of a site may help the scientist establish reasonable bounds on the minimum and maximum air temperature, the maximum change in air temperature over a given period of time, and how long air temperature is likely to remain constant. This information (which may change over time) can be used to flag certain values as missing (e.g., an impossibly large precipitation amount) or questionable (e.g., an excessively long period of zero wind speed).

(3) **Model-based gap filling**. Models can fill gaps (e.g., missing and questionable values) in the sensor data. Such models may be based, for example, on an empirical relationship between the missing parameter and other measured parameters. Often there are many possible models from which to choose, and scientist's choices may evolve over time.

In addition to allowing the scientist to retrieve or rederive earlier versions of the data, provenance can facilitate the development, testing, and application of these transformations by supporting the ability to undo and redo. For example, the use of models to fill gaps may require repeated adjustment, application, and evaluation of the model. By using provenance to undo part of a computation, the scientist can take advantage of some calculations without having to restart the analysis from scratch each time the model changes. Since this process may be repeated many times, the ability to back up and move forward, and to record all choices made so far, may represent a significant savings in time and effort.

The rest of this paper outlines our approach in Section 2, demonstrates how provenance can aid undo and redo on a scientific data-processing example in Section 3, places our work in terms of related research in Section 4, and summarizes our contributions in Section 5.

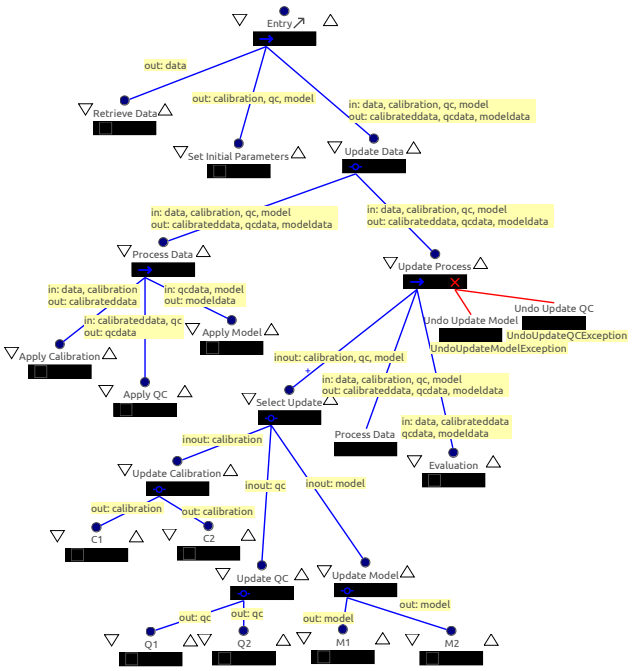

Figure 1: A scientific data-processing process definition, written using the Little-JIL process specification language [10]. In this process, a scientists collects data from sensors (`Retrieve Data`) and applies sensor calibration (`Apply Calibration`), quality control (`Apply QC`), and model-based gap filling to the data (`Apply Model`).

2 Our Undo and Redo Approach

To support scientists performing such data analysis, we have developed a system that allows users to undo sequences of data processing activities by reverting back to a previous state of the process execution. This allows the scientist to set aside previous processing results, revisit and change a previous decision, and move forward again, possibly automatically re-applying sequences of previously performed steps.

Using a detailed model of the process a scientists follow to process data (Figure 1), our approach tracks the history of the process execution as the scientist executes it. Since the process consists of steps, each of which modifies data artifacts, the state of the execution of the process at time t consists of the current step and the set of values of the artifacts. To revert to an previous point, the process has to revert both the control-flow and the artifact values. For example, when a scientist reverts back to the state of picking a gap-filling model, the expectation is that all data are restored to that point in the process, and the next step is to re-select a gap-filling model.

The detailed model of the process the scientist follows is critical to our approach, as is the Data Derivation Graph (DDG), the process-provenance structure that describes the history of the process execution [6]. The DDG records the history of all artifact creations and modifications, from which each historical execution state can be derived. When choosing to undo operations, our system presents the user with a visualization of the DDG (Figure 3) and enables the user to select the point to which to revert. Our approach requires explicit labeling of which steps are *revertible* (Figure 2) in the process specification. Thus, the operations that can be revisited must be defined *priori*. Theoretically, all steps can be revertible, allowing the user to undo arbitrary operations. After revisiting a step, subsequent operations can be reapplied automatically, either if they require no user inputs or by reusing the inputs from the previous execution, to the newly modified artifacts.

3 Scientific Data-Processing Case Study

We now illustrate our approach with a concrete example. As we mentioned in Section 2, our approach relies on a detailed definition of the process the scientist will follow in processing the data. Consider the following process (depicted graphically as we describe below, in Figure 1): The scientist collects raw data from sensors (we call this step `Retrieve Data`), and then chooses to either start to process the data using de-

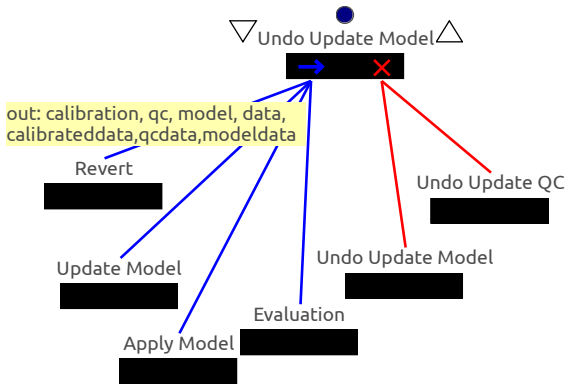

Figure 2: The refined definition of the `Undo Update Model` step from Figure 1. This exception handler guides the scientist to undo changes to `modeldata` and reapply a new model to `qcdata` that is recovered from an earlier point in the history.

fault parameters (`Process Data`), or update one of the parameters (`Update Process`) and reprocess the data (`Process Data`). The scientist evaluates the reprocessed data (`Evaluation`) and decides if anything needs to be undone and adjusted.

Figure 1 defines this process using the Little-JIL graphical process specification language [11]. Little-JIL has been used to support the definition of complex processes in the medical, election, software development, and other domains [4, 8, 12]. We describe only enough Little-JIL language to facilitate understanding of our examples, and refer the reader to previous work [11, 10] for to a detailed specification.

A Little-JIL process definition is a tree representation of the steps of a process, the artifacts created and modified in each step, and the resources required for each step. The tree diagram in Figure 1 is a hierarchical decomposition of the steps; each step denoted graphically by a black rectangular bar. The steps execute in the depth-first order, from left to right. A special *sequencing badges* denotes when a step's substeps (children) are to be executed sequentially (e.g., the arrow to the left of `Process Data` indicates that `Apply Calibration`, `Apply QC`, and `Apply Model` should execute sequentially), or one is to be chosen out of a set to be executed (e.g., the slashed circle sequencing badge on `Update Data` indicates a choice between `Process Data` and `Update Process`). Note there are two different `Process Data` steps. Processes may include recursion, referencing other parts of the tree, as, for example, `Process Data` does. Edge labels specify which artifacts may be accessed or modified. Red edges indicate exception-handling steps, which can alter the process' control flow, and which we use to implement our undo mechanism. For example, in

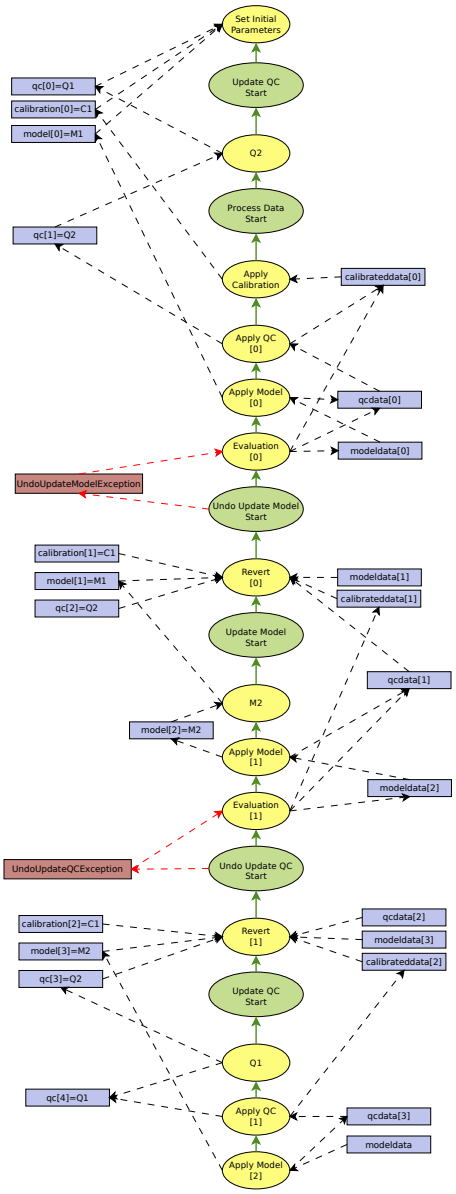

Figure 3: A DDG snippet shows the process execution history of a scientist reverting twice: to update the gap-filling model (`Apply Model`) and then again to update the quality control procedure parameter (`Apply QC`).

Figure 1, if during `Evaluation`, the scientist decides to undo the `Update Model` step, the system will throw an `UndoUpdateModel` exception. The execution control-flow will be brought to the `Undo Update Model` exception handler (see Figure 2).

Figure 2 details this `Undo Update Model` exception handler. The first leaf step, `Revert`, outputs all the artifacts that exist in the process, allowing setting aside changes and reverting to a previous state in the execution. (Note that in our implementation, the execution history

is immutable, so a *revert* creates a new copy of the old, reverted-to state, saving the history of all states.) The scientist can then revisit the `Update Model` step, and later be able to undo again, if desired, with a recursive exception handler call.

Figure 3 shows a snippet of the DDG generated by the process from Figure 1. In the DDG, ovals represent step execution instances (leaf steps are yellow and non-leaf steps green) and rectangles represent data instances. Exceptions (brown rectangles) are special data instances.

Here, the scientist has reverted twice. First, to re-update the gap-filling model, the scientist is directed to `UndoUpdateModel` step in response to `UndoUpdateModelException`, with `Revert` producing `qcdata[1]=qcdata[0]`. Once selected, the new gap-filling model (`model[2]`) is applied automatically to produce `modeldata[2]`. Second, to re-update the quality control procedure parameter, the scientist has two options: (1) Reverting to the point of starting `Apply QC[0]`, updating `qc` parameter, and proceeding with the newly selected quality control parameter and default `calibration[0]` and `model[0]` values. Or (2) reverting to the point of starting `Apply Model[1]`, updating `qc` parameter and proceeding with the default `calibration[0]` and the previously updated `model[2]` parameters. In this example, the scientist elects to use the earlier selection of `model` parameter, selecting `Apply Model[1]` in the DDG.

Our approach allows backtracking while keeping executed steps' provenance. We have implemented this capability as a command-line tool, which guides the scientist in performing data-processing tasks, allowing for revisiting and modifying earlier-made decisions.

4 Related Work

Provenance Map Orbiter [7] captures large provenance graphs and provides navigation mechanism using graph summarization and semantic zoom. Similar visualization mechanisms [1, 3] deliver the provenance information to the scientist using metadata queries. In contrast, our DDG is visual and takes advantage of Little-JIL's hierarchical structure.

Leeman proposed a formal approach to undo operations [5]. Some of the primitives he proposed are similar to ours. The notions of *undo list* to keep track of chronologically-ordered, program-state derivations and *time* to mark an event in the program, are similar to our proposed DDG and process control-flow definitions in the process domain. Rhyne and Wolf proposed adding a log of user actions, in addition to the history list that only keeps program state derivations [9]. This, like the DDG, joins control-flow and data-flow, but again, does not address the process. The selective undo model [2] allows the user to undo a number of operations, revisit a process step, and then automatically redo the other undone operations. Our model applies the redone operations to the modified artifacts, thus avoiding creating an inconsistent conflict between operations.

5 Contributions

We have developed a provenance-based approach for supporting undo and redo and scientific data processing. The approach uses provenance data, expressed as a DDG, to track the execution history of data processing and to allow scientists to explore that history to revisit, undo, and modify previous decisions. After revisiting a decision, our approach can guide the scientist in redoing previously executed steps in the new context. While a full, empirical evaluation of the benefits of our approach remains future work, our case study of a common, sensor-network, data-processing scenario shows promise that provenance-based support can reduce the cost of changing intermediate data-processing techniques in a complex, data-intensive process.

References

[1] M. K. Anand, S. Bowers, and B. Ludäscher. A navigation model for exploring scientific workflow provenance graphs. In *WORKS*, pages 2:1–2:10, 2009.

[2] T. Berlage. A selective undo mechanism for graphical user interfaces based on command objects. *ACM TCHI*, 1(3):269–294, 1994.

[3] O. Biton, S. Cohen-Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, pages 1072–1081, 2008.

[4] L. A. Clarke, Y. Chen, G. S. Avrunin, B. Chen, R. Cobleigh, K. Frederick, E. A. Henneman, and L. J. Osterweil. Process programming to support medical safety: A case study on blood transfusion. In *Unifying the Software Process Spectrum*, volume 3840, pages 347–359. 2006.

[5] G. B. Leeman, Jr. A formal approach to undo operations in programming languages. *ACM TPLS*, 8(1):50–87, 1986.

[6] B. Lerner, E. Boose, L. J. Osterweil, A. Ellison, and L. Clarke. Provenance and quality control in sensor networks. In *EIMC*, 2011.

[7] P. Macko and M. Seltzer. Provenance map orbiter: Interactive exploration of large provenance graphs. In *TaPP*, 2011.

[8] M. S. Raunak, B. Chen, A. Elssamadisy, L. A. Clarke, and L. J. Osterweil. Definition and analysis of election processes. In *ICSPSM*, pages 178–185, 2006.

[9] J. R. Rhyne and C. G. Wolf. Tools for supporting the collaborative process. In *UIST*, pages 161–170, 1992.

[10] A. Wise. Little-JIL 1.5 language report. Technical Report UM-CS-2006-51, University of Massachusetts, Amherst, 2006.

[11] X. Zhao, B. S. Lerner, L. J. Osterweil, E. R. Boose, and A. M. Ellison. Provenance support for rework. In *TaPP*, page 14, 2012.

[12] X. Zhao and L. J. Osterweil. An approach to modeling and supporting the rework process in refactoring. In *ICSSP*, pages 110–119, 2012.