RDataTracker

Collecting Provenance in an Interactive Scripting Environment

Barbara S. Lerner
Mount Holyoke College
blerner@mtholyoke.edu

Emery R. Boose
Harvard University
boose@fas.harvard.edu

Abstract

Scientific data provenance is often cited as a valuable tool for scientists to use to document their data collection and analysis processes, allowing improved understanding and sharing of data and results. However, most software that supports data provenance requires scientists to adopt new technologies rather than adding these capabilities to technologies that scientists already use. In this paper, we introduce RDataTracker, an R library that supports the collection of data provenance from executed R scripts.

Categories and Subject Descriptors J.2 [*Physical Sciences and Engineering*]

Keywords Scientific data provenance, R

1. Introduction

Scientific data provenance is cited as an essential requirement for documenting the work of scientists. Much as provenance in the art world helps authenticate the legitimacy of a piece of artwork, so, too, should scientific data provenance help authenticate the legitimacy of a scientific dataset or results derived from a dataset. Beyond this, scientific data provenance helps scientists understand their work and results, replicate the work of others, compare not just results but also scientific processes, and ensure the correctness of their data collection and analysis processes.

Currently, there are numerous systems that help scientists to collect, visualize, query and use their data provenance, including Kepler [2], Taverna [4], Vistrails [8] and Little-JIL [3, 5], among others. Unfortunately, these systems require the adoption of additional tools and techniques and may not work well with the software that scientists are accustomed to using. Even in cases where scientists can use their existing software, the software generally needs to be wrapped to become part of a larger workflow and the scientist needs to change his/her mode of interaction from working with the software directly to working with the larger workflow. While there are likely other advantages to working at the higher level of abstraction of workflow, it does introduce a technological barrier that many scientists are not interested in confronting.

The motivation for our work is to instead bring the collection of data provenance to the tools that scientists actively use. We are intentionally seeking as low a barrier to entry as possible, offering a minimal learning curve, while making data provenance useful and accessible to scientists. By doing so, our long term goals are to be able to study how creative scientists use data provenance if it is easy to collect, in particular, how they integrate the collection and analysis of data provenance into their work and what impacts data provenance have on the scientific results themselves. As a first step in this direction, we have been working on tools to support the collection, visualization, and querying of data provenance generated by the execution of R scripts and R commands entered interactively at the R console.

2. Background

R [6] is a language that has become very popular among scientists because of the strength of its support for statistical analysis and data visualization. Several features of R make it particularly attractive to scientists, such as vectorized functions and data structures like the data frame that allow scientists to easily work with tabular data. R also encourages data exploration by providing an interpretive environment that supports a combination of executing pre-existing scripts and entering of commands in an interactive console.

There are a few projects that have addressed the question of provenance for R users. R Markdown [1] is a tool that runs inside the RStudio environment (http://www.rstudio.com/). It allows a user to intermingle commentary with R code. When the user executes R Markdown code, it produces an html file as output that contains the user's commentary, the code that was executed and the output of the execution that was displayed on the console. The output of R Markdown is extremely useful for documenting what the script did, in a format that is suitable for education or for inclusion in a paper.

CXXR [7, 9] is an implementation of the R interpreter that collects provenance by placing hooks within the read-eval-print loop of the interpreter. The provenance is available to the user from within CXXR via two functions, provenance and pedigree. The provenance function reports the date and time a binding was created, the expression that was evaluated to produce its current value, and any expressions the current binding is used in. The pedigree function shows the history of expressions that were evaluated to arrive at the current value of a variable. However, these functions can only report information about the state of the environment at the point where the provenance and pedigree functions are called. In particular, it may not be possible to see the values that went into deriving the current value of a variable, if the variables used in the expressions involved in the derivation have been rebound. Furthermore, provenance is only collected at the top level, not from within function calls, and is not made persistent for later exploration.

Both of these efforts are valuable contributions to the collection of provenance for R. Our contribution is to collect a detailed provenance record, including intermediate values, as R executes, saving

this information in a provenance graph that is made available to the scientist for visualization, browsing and querying.

3. Approach

The lowest barrier to collecting provenance data would be to completely automate its collection, so that a scientist could run a script exactly as before, but with the result that the additional data provenance would also be recorded. In fact, this is the way that provenance collection works in many of the systems that support its collection. One downside of complete automation is that the resulting provenance data may be voluminous if it tracks the value of every variable used. In our work, we decided to approach this problem as a user of R, focusing our attention on collecting the data provenance that has the most value to the scientist.

Our approach relies upon two basic techniques. First, the scientist adds calls to functions defined in our RDataTracker library that collect data provenance as the script exectues. Second, the called functions use the capabilities that R provides for examining runtime state, including the call stack and variable bindings, to capture more detailed information. We thus seek a powerful payback for a small investment on the part of the scientist.

The data provenance is stored in a Data Derivation Graph (DDG) [5]. The nodes in a DDG are either procedural nodes or data nodes. Procedural nodes have several subtypes including ones to represent the start and end of procedures or blocks, and operational steps. Data nodes also have several subtypes including simple data, files, URLs, and error nodes. Procedural nodes are connected with control flow edges, while data nodes are connected to procedural nodes with input and output edges.

Figure 1 shows an R script that performs a standard processing sequence for electronic sensor data: reading data from a file, applying corrections to calibrate the sensors, performing quality control operations (such as removing outliers), and filling gaps where sensor data are missing (due to the removal of outliers, for example)[1]. This script includes a minimal amount of instrumentation. In particular, the scientist needs to include the RDataTracker library using the *source* function, initialize the provenance graph by calling the *ddg.init* function and save the provenance graph in a file by calling the *ddg.save* function at the end. Running this instrumented script creates a provenance graph, which is shown in part in Figure 2.

In the visualization in Figure 2, each yellow node represents an individual statement from the R script, while each lilac node represents a piece of data. Using our DDG Explorer tool, the user can view this visualization and by right-clicking on a data node see the value associated with that node. In this example, the data nodes labeled *11-raw.data* and *13-calibrated.data* hold data frames, while *12-calibration.parameters* holds the parameters used to perform calibration. The scientist could examine these data frames and the calibration parameters to see what effect calibration had on the raw data.

As this example shows, a minimal amount of instrumentation is able to save data provenance that may have considerable value. By investing a bit more effort in instrumentation, the scientist can further enhance the data provenance that are collected. This is done by adding additional calls to functions defined in RDataTracker. For example, the scientist may want to record the plots that are created by calls to the *plot.data* function. To do this, the scientist adds a call to the *ddg.procedure* function inside the *plot.data* function:

```
plot.data <- function (data)
  # Lines omitted that generate the plot
```

[1] For brevity, we have omitted the definitions of the functions *read.data*, *plot.data*, *calibrate*, *quality.control*, *gap.fill* and *write.result*, which also appear in the script file

```
source("ddg-library.r")
ddg.init("daily-solar-radiation.r", "ddg",
    enable.console=TRUE)

data.file <- "met-daily.csv"
start.date <- "2012-01-01"
end.date <- "2012-03-31"
variable <- "slrt"
raw.data <- read.data(data.file, start.date,
                      end.date, variable)
plot.data(raw.data,"R")

calibration.parameters <- read.csv("par-cal.csv")
calibrated.data <-
    calibrate(raw.data, calibration.parameters)
plot.data(calibrated.data,"C")

quality.control.parameters <-
    read.csv("par-qc.csv")
quality.controlled.data <-
    quality.control(calibrated.data,
                    quality.control.parameters)
plot.data(quality.controlled.data,"Q")

gap.fill.parameters <- read.csv("par-gf.csv")
gap.filled.data <-
    gap.fill(quality.controlled.data,
             gap.fill.parameters)
plot.data(gap.filled.data,"G")
write.result(gap.filled.data)

ddg.save()
```

Figure 1. Minimally instrumented R script

```
ddg.procedure(outs.file=list("filename"),
              lookup.ins=TRUE)
```

The *ddg.procedure* call instructs the provenance library to construct a node for the *plot.data* function. The name of the function and the input parameters are looked up dynamically by examining the call stack. This results in a data node being linked to the *plot.data* node to show the input. The output is identified by listing the names of the variables that hold the output values. In this case the value of the *filename* variable is *raw-plot.jpeg*. RDataTracker copies this file into a persistent area and creates a node to represent the file, linking it as output from the *plot.data* function. A relevant part of such a provenance graph is shown in Figure 3. The node's color indicates that it represents a file. The user can right-click on the node labeled *12-raw-plot.jpeg* to view the actual plot. A similar jpeg output node will be created after each call to *plot.data* in the original script. Thus, the user could compare the plotted data at different stages of processing in addition to comparing the values in a tabular format.

A long script may result in a provenance graph that is not well-suited for viewing in its entirety. To address this problem, RDataTracker allows the scientist to introduce levels of abstraction so that entire sections of the provenance graph can be collapsed to a single node. The scientist does this by adding calls to a pair of functions: *ddg.start* and *ddg.finish*, passing in a name for the abstract unit. This may be used, for example, to document the details of what happens within a function if the user wants to see those details. Figure 4 shows how the user might instrument the *read.data* function in the script to capture details of the initial

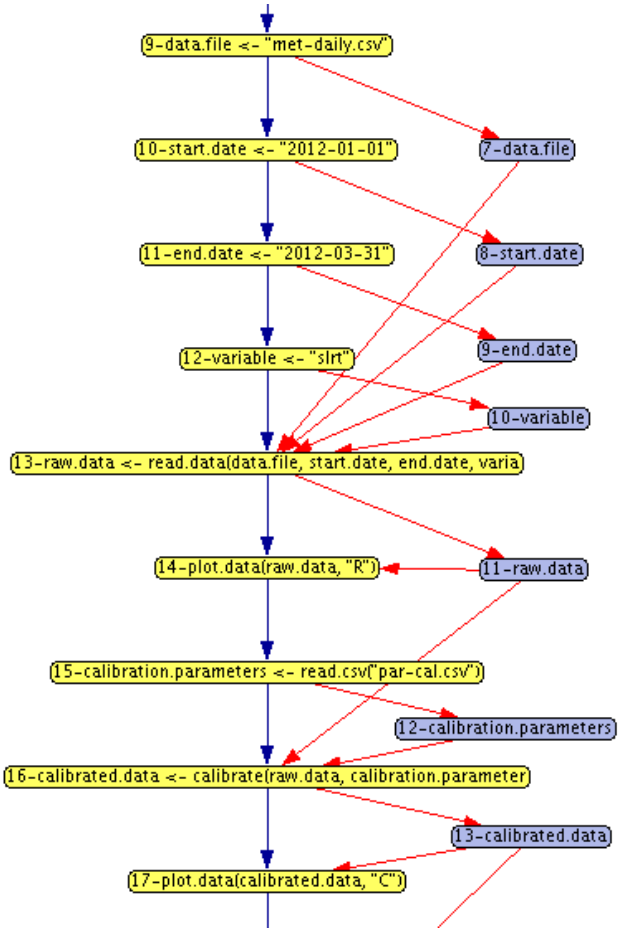

Figure 2. Provenance graph created by executing a minimally instrumented R script

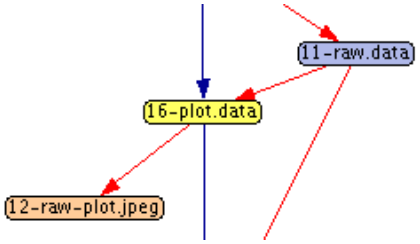

Figure 3. Provenance graph showing file output

data preparation. Here the user places calls to the *ddg.procedure* function to identify important actions within the *read.data* function, identifying the inputs and outputs of each of those sections.

Figure 5 shows how the provenance graph is displayed when the *read.data* provenance is collapsed to a single node, allowing the scientist to view *read.data* as a single abstract operation, while Figure 6 shows the detail that is visible when the node is expanded by clicking on it. Note that in the collapsed version, only data that flows into or out of the *read.data* function are shown in the visualization, whereas the expanded version shows the data flow within the function as well.

The *ddg.start* and *ddg.finish* calls can be placed anywhere and can be nested to any depth. If placed at the top level of the script,

```
read.data <- function(data.file,
                      start.date, end.date, variable)
  ddg.start("read.data")
  zz <- read.csv(data.file)
  ddg.procedure(pname="read.csv",
                ins=list("data.file"),
                outs.snapshot=list("zz"))

  zz$date <- as.Date(zz$date)
  all.data <<- subset(zz,
      zz$date>=start.date & zz$date<=end.date)
  ddg.procedure(pname="subset dates",
        ins=list("zz", "start.date", "end.date"),
        outs.snaphsot=list("all.data"))

  raw.data <<- all.data[c("date",variable)]
  names(raw.data)[names(raw.data)==variable]
      <<- "raw"
  ddg.procedure(pname="subset variable",
                ins=list("all.data", "variable"),
                outs.snapshot=list("raw.data"))

  raw.data$cal <<- 0
  raw.data$qc <<- 0
  raw.data$gf <<- 0
  ddg.procedure(pname="initialize data frame",
                ins=list("raw.data"),
                outs.snapshot=list("raw.data"))

  ddg.finish("read.data")
  return(raw.data)
```

Figure 4. Instrumentation to capture provenance within a function

they would allow grouping together operations without requiring the additional *ddg.procedure* calls that are needed when placed within a function.

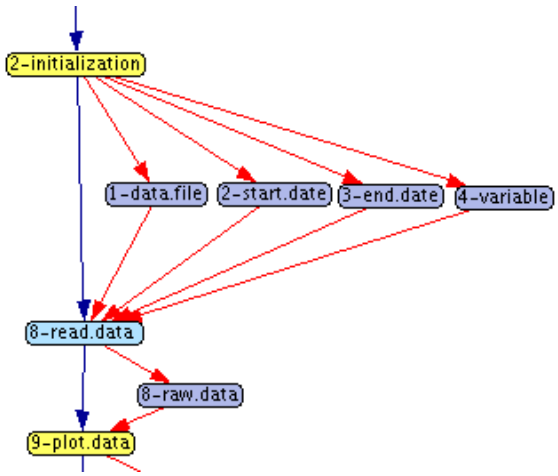

Figure 5. Abstracted provenance of the *read.data* function

4. Evaluation

We are at the early stages of encouraging scientists to use the provenance library on the scripts that they actively use. Early feedback

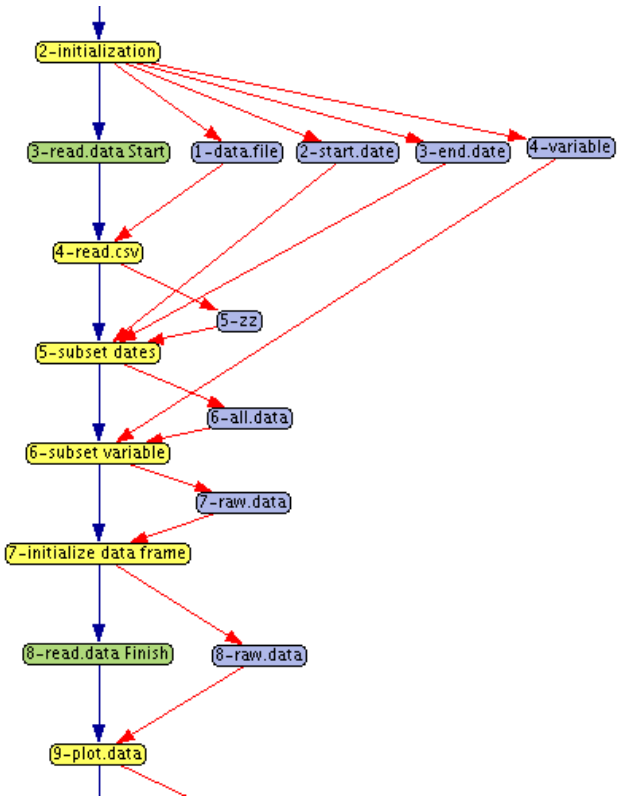

Figure 6. Detailed provenance of the *read.data* function

suggests that they find the provenance information that is collected to be quite valuable and that the minimal level of instrumentation shown in Figure 1 to be quite usable and understandable. Scientists are particularly appreciative of the ability to collect provenance without needing to learn to use a new tool. While not discussed in this paper, RDataTracker also records R run-time errors in the provenance graph, documenting execution up to the point of the error. This has also proven to be quite helpful for troubleshooting R scripts.

Scientists also report that the abstractions created by collapsible nodes are very helpful in managing larger provenance graphs and imposing a higher-level of understanding on the data collected. In fact, this has turned out to be quite useful as many scientists have limited training as computer programmers. As a result they often have scripts that are hundreds or thousands of lines long with almost no function declarations. The start and finish nodes impose abstraction on the provenance graphs in much the same way that a software engineer would introduce functions to make code more understandable.

Naturally, the more detailed instrumentation as shown in Figure 4 requires more time and may be prone to error. The provenance graphs depend on the names passed as the *ins* and *outs* parameters to *ddg.procedure* matching the names of variables in the script itself. Typos will result in incomplete graphs; calls that do not remain consistent with code will result in misleading provenance graphs. Simplifying the instrumentation effort is a major direction of our ongoing work.

One benefit of using instrumentation is that it helps address the problem of collecting so much data provenance that it become unmanageable. We expect that at different points in the lifecycle of developing an R script, the scientist may collect data provenance at different levels of detail, collecting more when a script is under development and less for long-term documentation, relying more on the concise source code as documentation. In fact, even within a single script, the level of provenance detail may vary, though the result will still appear to the scientist as a single provenance graph. With the current implementation, it will not be obvious if long stretches of the R script do no provenance collection. Balancing the simplicity of instrumentation, meeting the scientist's needs, remaining faithful to the script while avoiding a deluge of provenance data remain our primary goals.

5. Future work and conclusions

We are continuing our work of bringing provenance to scientists in a manner that allows them to work in a familiar environment, with minimal new technology to learn. We are currently involved in moving beyond our initial testing scripts to the scripts that scientists have developed for their daily work. By observing how scientists use RDataTracker and its companion tool, DDG Explorer, which provides the querying and visualization support, we expect to better understand the actual needs of scientists. We anticipate that when scientists have usable access to data provenance, they will develop new and interesting uses for it that we cannot foresee when looking at provenance from the perspective of a technology problem for computer scientists to solve.

Acknowledgments

The authors acknowledge intellectual contributions from collaborators Leon Osterweil and Aaron Ellison and Harvard Forest REU students Sophia Taskova, Antonia Oprescu, and Shaylyn Adams. The work was supported by NSF grants DEB-0620443, DEB-1237491, and DBI-1003938, the Charles Bullard Fellowship at Harvard University, and a faculty fellowship from Mount Holyoke College and is a contribution from the Harvard Forest Long-Term Ecological Research (LTER) program.

References

[1] B. Baumer, M. Cetinkaya-Rundel, A. Bray, L. Loi, and N. J. Horton. R markdown: Integrating a reproducible analysis tool into introductory statistics. *ArXiv e-prints*, February 2014.

[2] S. Bowers, T. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *IPAW 2006*, pages 133–147, Chicago IL, May 2006.

[3] B. Lerner, E. Boose, L. J. Osterweil, A. M. Ellison, and L. A. Clarke. Provenance and quality control in sensor networks. In *Proceedings of the Environmental Information Management (EIM) 2011 Conference*, Santa Barbara, California, September 2011.

[4] P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. Goble. Data lineage model for taverna workflows with lightweight annotation requirements. In *IPAW 2008*, pages 17–30, Salt Lake City, Utah, June 2008.

[5] L. J. Osterweil, L. A. Clakre, A. M. Ellison, E. R. Boose, R. Podorozhny, and A. Wise. Clear and precise specification of ecological data management processes and dataset provenance. *IEEE Transactions on Automation Science and Engineering*, 7(1):189–195, 2010.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org`.

[7] A. Runnalls and C. Silles. Provenance tracking in r. In *IPAW 2012*, pages 237–239, Berlin, 2012.

[8] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, 2008.

[9] C. A. Silles and A. R. Runnalls. Provenance-awareness in R. In *IPAW 2010*, pages 64–72, 2010.