Thank you very much for your thoughtful and detailed reviews of both the paper and the software. We have made improvements to both and noted some other areas for future improvement of the software.

We describe below how we have addressed each reviewer comment. In a few cases, we have decided to not make changes based on the comments and have explained why. Reviewer comments are indented with our responses following, unindented.

Reviewer 1 Paper comments

> * High level review
>
> This work represents exciting and powerful progress in the space of provenance capture and use in the R space. I have absolutely no doubt that both end-users and researchers in this space will find things in this work to be excited about. I certainly did. That said, there are a number of issues I would like to see addressed by the authors in terms of both the manuscript and the described software.
>
> I hope the authors find my review below helpful.

Your review has been extremely helpful! Thank you for all the detail and suggestions.

> * Overall Comments:
>
> All citations and bibliography are missing.

We are unsure why these were missing from the pdf. They should be present in the resubmission.

> In-text references (e.g. to figures/tables/etc in the article) are missing.

We are unsure why these were missing in the pdf. They should be present in the resubmission.

> Reproducibility materials (scripts) are missing.

We provided a script, but it seems to have been misnamed, which would explain why it was missing.

> * Missing section: limitations
>
> The systems described here are impressive, but not without limitations. Those limitations are not discussed in the manuscript. In my view a candid

discussion of scope and limitations of a work within the manuscript is both expected by academic norms and of great benefit to the reader.

Please add a limitations discussion and be upfront about the things that your systems are not currently able to handle.

This was an important omission on our part. We have added a Limitations section.

* Collecting provenance with rdtLite

** JSON format

1. missing citation for PROV-JSON standard

We have added a citation.

2. add Appendix/supplementary material which give concise example of their extensions to the standard

We have added a brief description of our extensions and a citation to a website that provides more information.

* Using Provenance

** "Provenance by itself has little value;"

this is not true as stated, I suggest rewording or removing this statement.

Provenance information is like the ore of a precious metal. It has little *direct* use *to the consumer*, but it has quite a bit of "intrinsic" value; one can always build new tools to exploit provenance - or any other - data once it is collected. If it wasn't collected, however, we are seriously limited in what we can do regardless of the tools we use (barring time machines).

You are exactly right. What you said is what we intended. We have reworded this.

* Related Work

** Missing Citations

The system closest in functionality to aspects of rdtLite is Becker et al's `trackr` and the underlying collection machinery, `histry`, which are quite similar to the collection done by rdtLite.

We have added this to the related work.

Temple Lang et al's `CodeDepends` does the type of input/output variable analysis flow graph modelling also done by rdtLite, and as of some time ago supports a wide range of NSE functions which aren’t currently handled correctly by rdtLite.

We have added this to the related work.

Reviewer 1 Software comments

* Software design/functionality

** Overall style

use of periods within function names other than as a prefix to indicate "hidden" is a pretty large violation of the R's idiomatic style becuase it is how S3 methods are indicated (i.e. `print.Date` is the `print` method for the `Date` S3 class).

I strongly recommend replacing periods with underscroes (_) in function names.

We appreciate your concern about the naming. The packages are currently on CRAN using dot notation. We have decided to stay with that naming for now but might move to snakecase in a future release.

** input/output tracing

*** NSE

rdtLite's input/output tracing does not correctly handle symbols used within calls to functions which utilize nonstandard evaluation (NSE)

To see this, replace

```

cars6Cyl.df <- allCars.df[allCars.df$cyl == 6,]
```

with

```

cars6Cyl.df <- subset(allCars.df, cyl == 6)
```

or with `dplyr::filter(allCars.df, cyl == 6)`), then do `prov.run` and

`prov.summarize()` on the modified script.

Doing so we get:

```

<snip>

PRE-EXISTING:

mtcars

cyl
```

```
<snip>
```

where, in fact, `cyl` is a column of mtcars, and not a separate pre-existing variable.

In fact, arguably, mtcars is not a pre-existing variable in the traditional sense either, as it is provided by the base package, making it a package symbol equivalent to functions provided by packages: the expression `data(mtcars)` does not use the value of `mtcars`, rather it uses `mtcars` as a symbol to be matched against the set of datsets `data` knows about.

Also note int he cases of both `cyl` and `mtcars`, they they are not displayed in the DDG Explorer as global inputs, which while correct by accident in terms of what is actually happening, is incorrect based on what prov.run/prov.summary *think* is happening.

These are great observations. You are correct that we do not handle non-standard evaluation. We have added discussion of this to the new Limitation section.

*** Sourcing child scripts

When script A sources script B directly, any global inputs of script B which weren't generated by script A (before the point where script B is sourced) are also global inputs of script A. This is not currently captured by rdtLite.

Correct. In order to capture the provenance inside a sourced script, the call to source must be replaced with a call to prov.source. This is mentioned in the section on rdtLite.

Given

./examples/bad_inner.R:

```

x <- x + 3
```

and

./examples/bad_outer.R:

```

source("./examples/bad_inner")
```

doing

```
x <- 5
prov.run("examples/bad_outer.R")
prov.summarize()
```

gives the following:

```
PROVENANCE SUMMARY for bad_outer.R

ENVIRONMENT:
Executed at 2022-02-28T16.59.47PST
Total execution time is 0.423 seconds
Script last modified at 2022-02-28T16.57.42PST
Executed with R version 4.1.2 (2021-11-01)
Executed on x86_64-apple-darwin17.0 (64-bit) running macOS Mojave
10.14.6
Provenance was collected with rdtLite 1.3
Provenance is stored in /private/var/folders/14/
z0rjkn8j0n5dj1lkdd4ng1600000gn/T/RtmpzdaKnO/
prov_bad_outer_2022-02-28T16.59.47PST
Hash algorithm is md5

LIBRARIES:
base 4.1.2
datasets 4.1.2
dplyr 1.0.7
ggplot2 3.3.5
graphics 4.1.2
grDevices 4.1.2
methods 4.1.2
provExplainR 1.1
provViz 1.0.8
rdtLite 1.3
```

stats 4.1.2

utils 4.1.2

vroom 1.5.6

SOURCED SCRIPTS:

/Users/gabrielbecker/gabe/checkedout/ArticleReviews/RJournal/making_provenance_work/examples/bad_inner.R

   2022-02-28T16.57.58PST

PRE-EXISTING:

None

INPUTS:

None

OUTPUTS:

File : /Users/gabrielbecker/gabe/checkedout/ArticleReviews/RJournal/making_provenance_work/dev.off.1.pdf

   2022-02-28 16:59:48

   6c64f02f67ac141c90e31191b0edf388

CONSOLE:

None

ERRORS:

None
```

** File detection

Input and output file detection appear to work, but are limitead by significant caveats not disclosed in the text.

Adding
```

write.csv(mtcars, "./mtcars.dat")

> mtcars <- read.csv("./mtcars.dat")
> ```
>
> near the beginning of the example script seems to result in the correct graph (as displayed by provVis::prov.visualize.run), HOWEVER, replacing the above with
> ```
>
> vroom::vroom_write(mtcars, "./mtcars.dat")
>
> mtcars <- vroom::vroom("./mtcars.dat")
>
> ```
>
> does not. `vroom_write` is not recognized as writing a file. This is, of course, a toy example with the read and write statements directly next to eachother, but in a realistic example, any functionality which relies on the ability to track lineage across write and read cycles will be broken in ways that I suspect will not be fun for the user to try and debug.

We have addressed this in two ways. First, the software has been updated to recognize the I/O functions in the vroom package. Second, we have added discussion of the limitations of I/O tracking to the Limitations section.

> Also, `pkgname::funname(stuff)` appears to display as "package.rds" being an "input file" to the expression, but library(pkgname) does not, so one of these is incorrect (my suspicion is that the library behavior is intention and the `::` is not, but cases could be made for either (though not for being inconsistant as is the case now, in my opinion).

Neither the library function nor :: syntax result in package.rds showing as an input file. There may be other things, such as version checks in the R code that cause the package.rds file to show up and this is confusing. We have modified the labeling on nodes that reference the package.rds file to now show what package the package.rds file comes from in order to provide more helpful information.

> ** Lineage Begins with Library
>
> When retrieving the backwards lineage of a value, the `library` calls required to have the symbols used in expressions are not included. This means that this lineage is not "runnable as is" in a clean session.
>
> I would like to see this remedied so that the expression lineage is a true tree-shaken script that can be exported and run on its own if desired.

This is a great idea and something we plan to include in a future release.

** DDG Explorer

*** `data` calls

1. `data(mtcars)` causes 3 input file nodes to appear, which is incorrect

You should just see the datasets node as input now. If you run it a second time, you will see 2 nodes, the datasets node and an mtcars node, as it now sees mtcars as a predefined variable (as mentioned above in the NSE discussion).

2. right-click -> "show how variable was computed" on input file node
   causes java exception to appear in console

This bug is now fixed. A message appears indicating this is an external input.

** provExplainR

*** Detecting differences in source code

The difference detection used by provExplainR to detect differences in source code is (arguably) overly sensitive in a number of ways:

It operates on raw text not parsed expressions. This means that whitespace and other fully execution-equivalent changes will trigger messaging to the end user that the script has changed.

As an illustrative, if corner-case, example, consider that

```
identical(parse(text = "sup('yo')", keep.source = FALSE),
          parse(text = 'sup("yo")', keep.source = FALSE))
```

returns `TRUE`, meaning that while the files were different, the *instructions to the computer were not*.

Great observation! We use a standard textual difference strategy like what is commonly used in IDEs. Detecting differences based on parse trees would be more accurate and we will investigate that for a future release.

*** Default script for comparison

The code is saved out in the provenance directories, so giving two provenance directories should be sufficient; users should not be required to also pass a file.

Default to having the code for comparison to the code captured in the first one.

Thank you for this suggestion. We made this change.

Reviewer 2 Paper comments

Overview

The authors present an interesting set of tools to analyze both coarse and fine provenance data of R programs. The tools are user-friendly and the resulting data is presented in a way that can be easily understood. Methods are up-to date. I believe that, as the R ecosystem increases its scope of application and range of users, such tools are valuable since they facilitate reproducible analysis and identification of sources of error and differences between executions. However, I have found major issues when attempting to use the tools with a set of test cases, which I believe should be resolved.

Article

The paper is well written and I could easily understand the concept of provenance and its relevance. However, I believe the order of the presented examples might be slightly re-arranged to facilitate the understanding of the overall procedure. For example, in section “A first example”, tools from both the base rdtLite and the provDebugR package are used (from the latter, debug.lineage). Then section “The end-to-end provenance tools” introduces the concept of multiple related packages operating at different levels, and following subsections present functionalities of each of the packages. Then, for example, in subsection for provDebugR, the debug.lineage function is presented again towards the end of this subsection.

I believe it might be easier to follow for general users if first the idea that we are provided with a set of related packages would be presented first, and then following subsections described the functionalities provided in each package. Additionally, it would be nice to have a summary of which are the main functions provided by each package at the beginning of each of these subsections.

Thank you for this suggestion. We have taken it into consideration and have respectfully decided to stick with the original ordering. We feel that the first example serves as motivation that will draw the reader into wanting to know more about the tools. Then, before diving into the details of the tools, we present the concept of it being implemented as a collection of packages. We show examples of the major functions of each package, but feel that starting with a summary would make the paper feel too much like a user manual or help pages.

Also, I think it would be nice to provide some basic examples of the developer tools included in provParseR and provGraphR.

The paper does provides some examples. The provParseR package basically provides a number of getter methods that allow exploration of the collected provenance, and several of these are described. The provGraphR package really only has one function, which is the get.lineage function described in the paper, and that provDebugR uses extensively.

> About the discussion section, I believe it would be nice to briefly reference some of the main tools available for provenance analysis in other languages too, such as Python. Also, the trackr package, available on CRAN, seems related. It would be nice to briefly mention the differences in implementation or purpose with this package.

We have updated the related work section to include trackr and to highlight some provenance collection tools for other languages.

> As minor comments, there seems to be some formatting issue with the Bibliography and references to display items such as figures and listings, which are referenced as question marks in the in-text citations.

Thank you for pointing this out. There was apparently a problem with the pdf submitted and this should be fixed now.

Reviewer 2 Software comments

> The code is well-written and documented, and includes numerous comments which facilitate understanding of the logics of each function. However, I have found errors when attempting to use the described functionalities with my own test cases. I have attached a main script (tests.R) and the corresponding required set of testing scripts, with comments that describe the errors. Also, I was not able to run prov.visualize , even in the context of the provided mtcars_example.R script.

Thanks for sharing your test script with us. It was fun using space station data. You found some bugs in our code that have been fixed. Specifically, error cases 1, 2 and 3 were all caused by the same problem. Your code calls read.table and our code was expecting the table to be coming from a file, but it is coming from a text string. We now check for this case.

provViz now checks if Java is available on the user's computer and gives a message if it is not installed: "Java must be installed in order to visualize the provenance".

You also identified an NumberFormatException error in provViz. This was an internationalization problem. The JSON parser in provViz assumed . would be used as the decimal marker, and in your case , was used. We have improved the parser to be locale-specific.

You note that provExplainR did not find the difference in the script when you modified it. However, the provExplainR output says:

```
The content of the main script has changed
Run prov.diff.script to see the changes.
```

Running prov.diff.script highlights the changes that you have described. We do recognize that the output from provExplainR could be improved to make it easier to read and we plan to do that in a future release.