Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 637–647

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

A Universal Identifier for Computational Results

Matan Gavish and David Donoho

Stanford University

Abstract

We present a discipline for *verifiable* computational scientific research. Our discipline revolves around three simple new concepts – *verifiable computational result* (VCR), *VCR repository* and *Verifiable Result Identifier* (VRI). These are web- and cloud-computing oriented concepts, which exploit today's web infrastructure to achieve standard, simple and automatic reproducibility in computational scientific research. The VCR discipline requires very slight modifications to the way researchers already conduct their computational research and authoring, and to the way publishers manage their content. In return, the discipline marks a significant step towards delivering on the long-anticipated promises of making scientific computation truly reproducible.

A researcher practicing this discipline in everyday work produces computational scripts and word processor files that look very much like those they already produce today, but in which a few lines change very subtly and naturally. Those scripts produce a stream of verifiable results, which are the same tables, figures, charts and datasets the researcher traditionally would have produced, but which are watermarked for permanent identification by a VRI, and are automatically and permanently stored in a VCR repository. In a scientific community practicing Verifiable Computational Research, exchange of both ideas and data involves exchanging result identifiers – VRIs – rather than exchanging files. These identifiers are controlled, trusted and automatically generated strings that point to publicly available result as it was originally created by the computational process itself. When a verifiable result is included in a publication, its identifier can be used by any reader with a web browser to locate, browse and, where appropriate, re-execute the computation that produced the result. Journal readers can therefore scrutinize, dispute, understand and eventually trust these computational results, all to an extent impossible through textual explanations that constitute the core of scientific publications to date. In addition, the result identifier can be used by subsequent computations to locate and retrieve both the published result (in graphical or numerical form) and the original datasets used by its generating computation. Colleagues can thus cite and import data into their own computations, just as traditional publications allow them to cite and import ideas.

We describe an existing software implementation of the Verifiable Computational Research discipline, and argue that it solves many of the crucial problems commonly facing computer-based and computer-aided research in various scientific fields. Our system is secure, naturally adapted to large-scale and cloud computations and to modern massive data analysis, yet places effectively no additional workload on either the researcher or the publisher.

Keywords: reproducible research, verifiable computational research, verifiable result, verifiable result identifier, computation chronicle, VCR repository

doi:10.1016/j.procs.2011.04.067

1. The current situation, and our response

Many articles in modern science have the following structure: there is some expository text, and there may be expository figures that are rendered by an artist, and then there are the real results: figures, tables and charts that have been prepared by computer data analysis scripts. The significance of the article completely stands or falls based on these *computational results* and our interpretation of the computer program that created those results.

The prevalent workflow for the creation and publication of computational results, practiced in computer-based or computer-aided research, is as follows.

Current workflow in computational science

1. Store a private copy of the original data to be processed on the local machine.
2. Write a script or computer program, with hard-coded tuning parameters, to load the data from the local file, analyze it, and output selected results (a few graphical figures, tables, etc) to the screen or to local files.
3. Withhold the source code that was executed, and the copy of the original data that was used, and keep them in the local file system in a directory called e.g. "code-final".
4. Copy and paste the results into a publication manuscript containing a textual description of the computational process that presumably took place.
5. Submit the manuscript for publication as a package containing the word processor source file and the graphical figure files.

Researchers from numerous scientific disciplines have expressed the opinion that this workflow is highly problematic for scientific research. Although this opinion has been expressed in publications dating back two decades, the problem seems to be growing more acute. In recent very visible examples [1, 2] identified medical research affecting treatment of human subjects, and showed that this work simply was not reproducible, namely that the scientific basis for the treatment allocations could not be reproduced. Another recent example is the debate concerning the policy-shaping climate change research [3]. Many authors have by now advocated the practice of *Reproducible Computational Research* [4, 5, 6, 7, 8, 9, 10]; similar concerns motivate research on Computational Provenance [11, 12], which is receiving increasing attention as the problematics become more visible [13, 14, 15].

Advocates of "Reproducible Research" typically propose to adopt a more ambitious workflow.

Additional Workflow for Reproducible Computational Reserch

1. Manage source code and data in a version control system, instead of standard local files.
2. Publish along with the manuscript, a "code-data dump": a folder containing all program source code and data files used.
3. Publish, along with the manuscript, a "makefile": a script that generates all figures included in the publication.

Unfortunately, although such gestures head in the right direction, they have not become main-stream research practices. Indeed, each solution proposed is idiosyncratic – depends heavily on local hardware and software – and therefore inherently can address only small communities of users. Furthermore, proposed solutions usually impose significant additional workload (e.g. creation of a makefile after the experiment is over), hence used only by those dedicated to the reproducibility cause. *No solution has yet been proposed that is disciplined, standard, simple and automatic.*

In this paper we introduce three simple, general notions which, properly used, allow the easy practice of reproducible research and publishing of reproducible, or even"executable" papers. With these notions in hand, a broad range of scientific researchers, their communities and their journals can easily and naturally begin to practice fully reproducible research. The notions are:

- *Verifiable Computational Result* (VCR). A computational result (eg. table, figure, chart, dataset), together with the metadata describing in detail the computations that created it (details below).
- *Verifiable Result Repository* (Repository). A web-services provider that archives VCRs and later serves up views of specific computational results (details below).
- *Verifiable Result Identifier* (VRI) A URL (web address) that universally and permanently identifies a repository and causes it to serve up views of a specific VCR (details below).

These notions are very natural within the modern viewpoint of web services and cloud computing.

To apply these notions, researchers work within their traditional workflow in scientific computing, effectively changing a few keywords in the same scripts they would have been creating anyway. Authors work within their traditional word processor, creating the same documents they would have created anyway, but effectively changing very slightly the way they link documents to figures and tables. Publishers publish the same online journals they would anyway, within their traditional content management schemes, but add hyperlink capability to any result, linking to new content types offered by the repository servers they operate.

While these changes in workflow are minimal, as we will see, the impacts they create – in facilitating scientific transparency, reproducibility and exchange of ideas through publications – are far-reaching.

2. The Verifiable Computational Research discipline

VCR is a discipline that at the same time allows researchers to easily create verifiable results, and forces them to do so. This is possible thanks to a VCR software system, working transparently in the background of the computation and communicating with a VCR repository. In this section, we describe the abstract principles of VCR. The next sections explain how computations are chronicled, and outline VCR system design, implementation and usage.

The Three Principles of VCR

1. *Computation means publication:* In Verifiable Computational Research, every computation automatically generates a detailed *chronicle* of its inputs and outputs as part of the process execution. The chronicle is automatically stored in a standard format on a VCR repository for later access.
2. *VRI for every result:* In Verifiable Computational Research, every figure, table, chart, and dataset is watermarked by a Verifiable Result Identifier (VRI), a DOI-like string that permanently and uniquely identifies the chronicle associated to that result and the repository that can serve views of that chronicle. VRIs are
   - *Timely:* The VRI is created when the result is generated – never retroactively.
   - *Characteristic:* Results created under the exact same conditions will carry the same VRI.
3. *VRI-based communication:* All communication among researchers and between their computations is handled by exchanging existing published VRIs – not files.

Practice of the VCR principles leads to a workflow subtly different from the prevailing workflow, yet filled with long-term benefits not previously available.

The Verifiable Computational Research workflow

1. Obtain the VRI of the dataset to be processed. If the VRI does not yet exist, submit the data to a repository and obtain a VRI.
2. Write a script or computer program to obtain the data via its VRI, analyze it, and create results (graphical figures, tables, etc).
3. Before executing the program, commit to a VCR repository that the intended audience of the research trust. (In day to day work, commit to a personal or research group repository; When ready to submit to a journal, commit to that journal's VCR repository; When creating a grant application, commit to the funding agency's VCR repository; and so on.)
4. During the program execution, a detailed chronicle of the computation is automatically published in the specified repository. Every result is automatically branded with a VRI.
5. Obtain feedback from the repository (e.g an email) with a list of VRIs generated during the computation. Access the original data, computation chronicle and results by directing a web browser to these VRIs.
6. Copy and paste the result VRI into the publication manuscript source file (e.g. word processor file or TeX source file).
7. Submit the manuscript for publication as a single source file – *without including or attaching any other file.*

3. The Components of a VCR System

The VCR system is able to record computational processes at run-time, and then to validate, archive, serve and sometimes re-execute them. The component installed on the researcher's machine consists of (1) a computational environment extension, able to record computation chronicles, and (2) a word processor extension. The VCR repository is the central component installed on the group, publisher or institute machines. We now describe each.

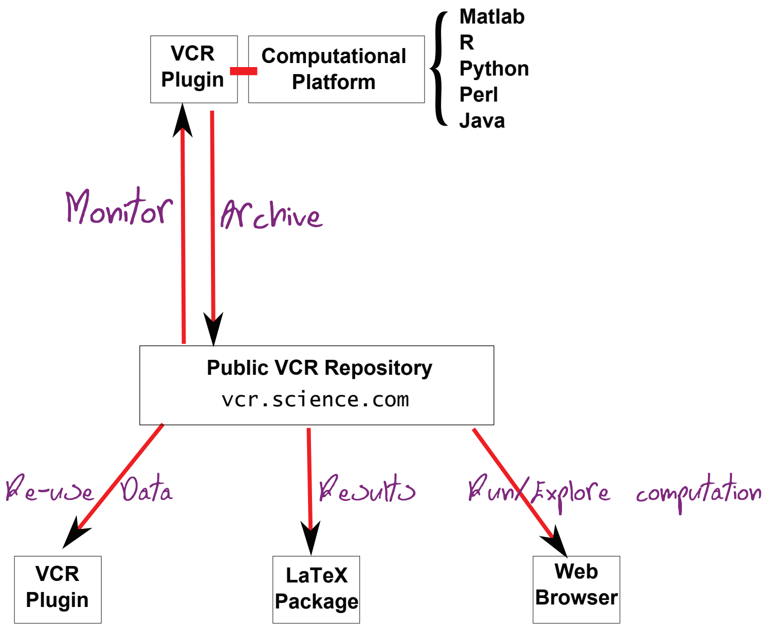


Figure 1: The components of a VCR system

3.1. The VCR Environment Extension and the Computation Chronicle

Most scientific computing environments in use today, including Matlab, Python, Scilab, R, S, Stata, Perl, Ruby, Java and numerous other adhere to the procedural programming paradigm [1]. In procedural programming, the main operation is invoking functions, which themselves invoke functions and so on. The course of an entire computation thread can be thought of as a tree of function invocations, whose root is the invocation of the main function, and whose leaves are invocations of functions that did not themselves invoke other functions. A function is invoked with input parameters – in our context, data and tuning parameters – and returns output parameters. Archiving the entire course of the computational process is equivalent to recording the computation tree of function invocations, similar to what a debugger would do, and *chronicling* each invocation, namely, recording their input parameters received at run-time, code executed, and output parameters returned at run-time.

In a computer-based experiment or data analysis, the invocation tree is mostly composed of platform-standard service functions, which are of little interest. The relevant part of the computational process should allow those scrutinizing the computation to understand exactly how any VCR was generated. This includes:

1. The function invocations that generated Verifiable Results (VCRs).
2. The function invocations that directly or indirectly invoked the above.

These *"chronicled invocations"* populate a small sub-tree of the whole invocation tree. To archive the relevant part of the computation we thus need to record the following on a computation repository:

1. Chronicle (code, input and output) of all chronicled invocations, namely all invocations that eventually lead to generation of VCRs.
2. All VCRs generated by the process.
3. All dependencies – source code of user-defined functions or binary software packages – used by the process, which are not documented environment-standard.
4. All screen messages echoed during the process.
5. Meta-data such as platform name and version, author, copyright information and archiving time.

The VCR Computation Environment Extension (VCR plugin, for short) is a small piece of software, developed specifically for some computing environment. It adds reserved words that allow the researcher to connect to a VCR repository, perform chronicled invocations and declare some variables as VCRs. Behind the scenes, it automatically handles the complicated task of recording the computation chronicle and communicating it in standard – not environment specific – format to the VCR repository.

3.2. The VCR Word-Processor Extension

In the VCR discipline, only VCRs – results that were issued a Verifiable Result Identifier – may be included in publications. These results exist on a VCR repository, not as local graphic and data files. The VCR Word-Processor Extension add functionality to word processor used for scientific publications - notably, LaTeX and Word – and allows to include a result specified by its VRI rather than by including a local file.

3.3. The VCR repository

The VCR repository is a stand-alone web server and database, to which VCR plugins transmit computation chronicles. A VCR repository archives results indefinitely, issue VRIs, verify the consistency of chronicles, and serve views of results it stores. The views include dynamic web pages, created when readers direct their web browsers to VRIs, or specifically formatted information, such as dataset requested by a VCR plugin or a rendering of graphical result requested by a word-processor extension when compiling a document. While the details are beyond the current scope, a VCR system include a natural way to cope with private datasets that cannot be submitted to a public repository, protect intellectual property of code submitted to a public repository, and deal with computations that uses proprietary software packages.

[1]Fully Object Oriented software design is rarely practiced in scientific computing, simulation or data analysis.

3.4. The VRI

The Verifiable Result Identifier is a cryptographically secure digital signature that encodes all information about the creation conditions of the result.

3.5. A note on short and long-term Re-executability

Under certain conditions, the VCR repository is able to accept a request to re-execute a computation whose chronicle it stores. The VCR computation chronicles allow re-execution of processes under the same conditions in which they were executed at the original run-time. Inevitably, re-execution requires that the computing environment originally used still be available and licensed on the repository – a tremendous difficulty for perpetuating publications. In fact, the short history of electronic computers tells us that this might be impossible: in all likelihood, *any computing platform will become extinct* – obsolete and unavailable anywhere on earth – *within no more than a few decades* . (If you disagree, try re-executing a computer-based experiment from the 1970's, such as a statistical analysis that used the MISS data analysis package for PDP11 minicomputer [16]).

Fortunately, long-term executability is partially unnecessary. When scrutinizing a computer-based experiment or data analysis, and when trying to understand the reasoning that lead to the published results, we rarely want to re-execute the original program. Readable source code, its dependencies, and the actual run-time values of input and output parameters of chronicled functions are what we really need in order to understand how computational results were generated. These are perpetuated in our computation chronicle and, unlike the computing environment used, never expire.

In other words, while we do make re-execution of archived processes possible for as long as their computing environment is available, and certainly for the short term, we suggest that platform-free browsing of the archived process and computation tree may be more valuable, and more realistic, than a perpetual ability to re-execute whole programs.

4. Existing VCR Software Implementation

The VCR system we describe has already been implemented in software. VCR plugins has been developed for the popular Matlab, R and Python environments on all major operating systems. A VCR word-processor extension is being developed for both Word and LaTeX word processors. A VCR repository has been developed and deployed on the Stanford University Statistics Department computers, and is in pilot use.

A list of available VCR plugins, as well as a list of publicly available VCR repositories and a tutorial on VCR will be available at the time of publication at `http://vcr.stanford.edu`.

We now proceed to describe the usage of our VCR implementation by the researcher, the publisher and the reader.

4.1. The researcher workflow: Producing Verifiable Computational Results

To create VCR's, the researcher installs a the VCR Computation Environment Extension ("plugin") to their computing environment of choice. As explained above, the plugin allows declaration of variables as VCRs and automates chronicling of selected function invocations, namely, recording the input parameters received, code executed, and output parameters returned.

Our Matlab, Python and R plugins are very similar in appearance. They introduce the reserved words `verifiable` and `chronicled`, and the commands `repository` and `loadvcr`. Adding `verifiable` to a variable definition assigns a VRI to that variable, turning it into a VCR. Adding `chronicled` to a function invocation records that invocation in the way described above. The command `loadvcr` imports a VCR generated by a previous computation into the workspace, similarly to a file-load command. At the beginning of the computation, the process commits to the VCR repository – which will monitor and archive the computation – that is specified by the `repository` command. The actual repository used (individual, group, institutional or journal repository) should depend on the intended usage of the VCRs to be produced.

As a simple example, here is a Matlab program that studies image inpainting. The steps of the experiment are: (1) load an original image, (2) degrade it artificially, (3) attempt to recover the image using the inpainting method under consideration. Without using VCR, the business part of Matlab program reads:

```
function inpaint_experiment_main
    original = load('C:\MRI.jpg');
    corrupt = corrupt_image(original);
    recover_image(corrupt);
end function main

function recovered = recover_image(corrupt_image)
    recovered = % ... image inpainting code
    figure1 = imageplot(recovered);
end function
```

With the VCR plugin, very little changes are needed in order to archive a detailed computation chronicle on a VCR repository, and assign VRIs to the figures created:

```
function inpaint_experiment_main
    repository('vcr-aware-journal.com',username='matan')
    original = loadvcr(other-journal.com/371aee2f-0d1f-405b-f1dd-7d4446363324/format=matlab);
    chronicled corrupt = corrupt_image(original);
    chronicled recover_image(corrupt);
end function main

function recovered = recover_image(corrupt_image)
    recovered = % ... image inpainting code
    verifiable figure1 = imageplot(recovered);
end function
```

Behind the scenes, the VCR system is busy: The repository server at `vcr-aware-journal.com` is contacted; The archived VCR containing the original image to be processed is downloaded from the repository at `other_journal.com` and formatted; A VRI is assigned to the VCR `figure1`. Finally, when the process terminates, the chronicle it is automatically generated, encrypted and communicated to the repository at `vcr-aware-journal.com`. A receipt is emailed by the repository:

```
Matan, Hello from the computation repository at vcr-aware-journal.com
Your process has been uploaded on 10/12/2009 20:17:58 PST. Here is your receipt:

Verifiable Result                   VRI: cd6e83d7-3929-4452-9259-fffcde56547b
"figure1"
                                    http://vcr-aware-journal.com/cd6e83d7-3929-4452-9259-fffcde56547
```

When using an established public repository, the VRI (which is a digital signature) and server timestamp provides the author with a proof of code/data ownership.

From this point on, the process archived on the repository is universally and permanently connected to `figure1`. The VCR plugin makes sure that the researcher cannot declare a certain result Verifiable, while withholding the data and code that generated it. Furthermore, in accord with the VCR principles, the deliverable product of this computation is a VRI - not local graphics or data files.

4.2. The author workflow: Composing and submitting a publication with VCR's

In order to include "figure1" in a LaTeX manuscript (say) submitted for publication, the author (1) makes sure that the manuscript is submitted to the journal whose repository has been used, and (2) installs the VCR Word-processor extension for LaTeX. This extension include a VCR into any document or presentation, directly from its owner repository. For example:

```
% LaTeX source:
\usepackage{vcr}
\repository[vcr-aware-journal.com,username=matan]
...
\includeVCR[width=10cm]{cd6e83d7-3929-4452-9259-fffcde56547b}
```

Note that conveniently, the nature of the VCR (figure, table, etc) and any graphical format issue is no longer of the author's concern. When the document is compiled, the VCR repository is automatically contacted, and produces the VCR in the format specified by the LaTeX document specification.

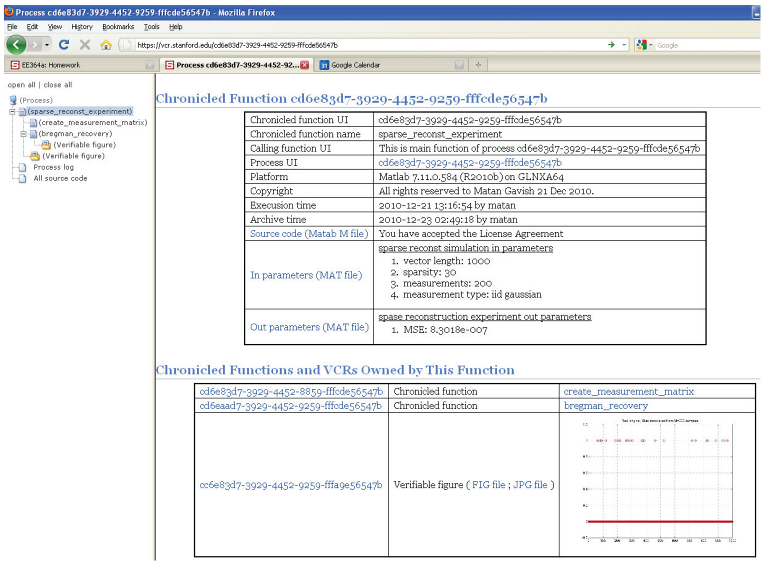
For graphical VCR's, the repository watermarks the VRI both in both human-readable text and machine-readable QR barcode. The image becomes a hyperfigure (if you are reading this in an online version, try clicking on Figure 3 below). For numerical VCR's, the VRI is added in a footnote and the numeric value is a hypertext linking to the computation repository, like so: $p = 0.07$[2]. This turns each VRI in a publication or presentation into an entry point into the generating computation.

Therefore, the TeX source submitted for publication either contains authentic, valid VCR's that already exist on the journal's repository, or simply would not compile.

4.3. The reader workflow: Browsing, re-executing and importing generating computation of a published VCR

Suppose that `figure1` has been included in a paper published by the journal "VCR-Aware". Ten years after the publication, a reader would like to scrutinize `figure1`. The watermark at the lower-right corner of the figure at once indicates the VRI, the owner repository and the access URL (Figure 3).

In an online publication, a click on the hyperfigure leads to this URL. In a paper publication or ongoing presentation, the reader types the access URL into a web browser or uses any optical scanner to read the barcode. Our repository software now acts as a secure web server: if the reader has subscription privileges, *he or she can now browse the computation chronicle using any web browser*, as well as the code and data used in the computation. The values of input and output parameters at any chronicled function invocation allow careful scrutiny of the computation generating the VCR, even if the original platform has become extinct. If the dataset used was generated as a VCR of a previous computation, it can be traced to its origin across repositories and publications.



Process cd6e83d7-3929-4452-9259-fffcde56547b - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://vcr.stanford.edu/cd6e83d7-3929-4452-9259-fffcde56547b
Google
EE364a: Homework
Process cd6e83d7-3929-4452-92...
Google Calendar
open all | close all
(Process)
(sparse_reconst_experiment)
(create_measurement_matrix)
(bregman_recovery)
(Verifiable figure)
(Verifiable figure)
Process log
All source code

Chronicled Function cd6e83d7-3929-4452-9259-fffcde56547b

Chronicled function UI	cd6e83d7-3929-4452-9259-fffcde56547b
Chronicled function name	sparse_reconst_experiment
Calling function UI	This is main function of process cd6e83d7-3929-4452-9259-fffcde56547b
Process UI	cd6e83d7-3929-4452-9259-fffcde56547b
Platform	Matlab 7.11.0.584 (R2010b) on GLNXA64
Copyright	All rights reserved to Matan Gavish 21 Dec 2010.
Execusion time	2010-12-21 13:16:54 by matan
Archive time	2010-12-23 02:49:18 by matan
Source code (Matab M file)	You have accepted the License Agreement
In parameters (MAT file)	sparse reconst simulation in parameters 1. vector length: 1000 2. sparsity: 30 3. measurements: 200 4. measurement type: iid gaussian
Out parameters (MAT file)	spase reconstruction experiment out parameters 1. MSE: 8.3018e-007

Chronicled Functions and VCRs Owned by This Function

cd6e83d7-3929-4452-8859-fffcde56547b	Chronicled function	create_measurement_matrix
cd6eaad7-3929-4452-9259-fffcde56547b	Chronicled function	bregman_recovery
cc6e83d7-3929-4452-9259-fffa9e56547b	Verifiable figure (FIG file ; JPG file)	

Figure 2: Browsing a computation chronicle using a Firefox web browser - a screenshot

[2]VRI: repository.journal.com/cd6e83d7-3929-4452-9259-fffcde56547b

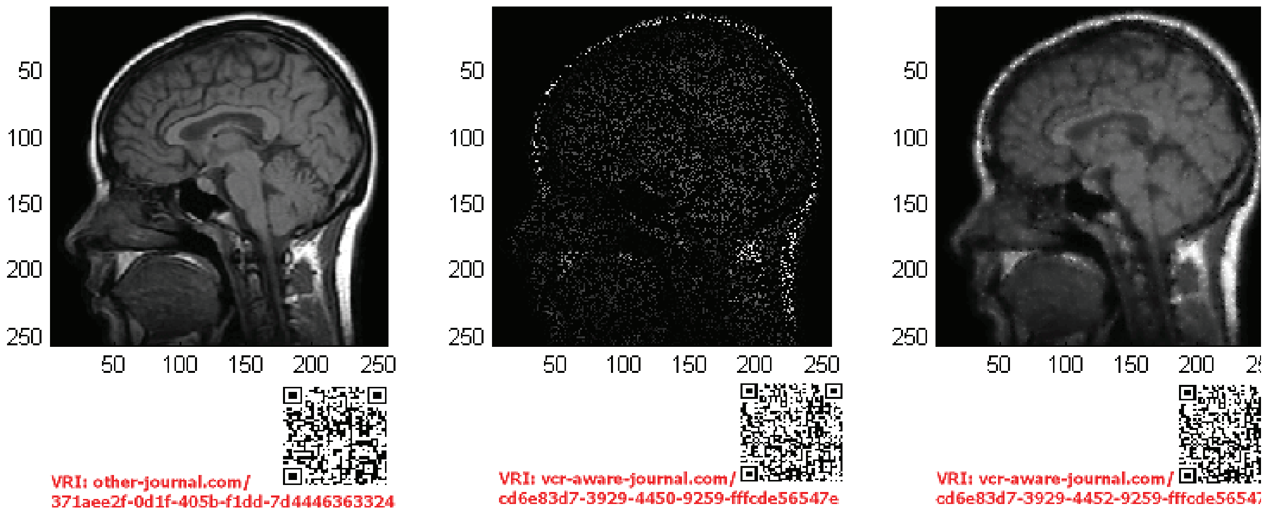

50
100
150
200
250
VRI: other-journal.com/
371aee2f-0d1f-405b-f1dd-7d4446363324
VRI: vcr-aware-journal.com/
cd6e83d7-3929-4450-9259-fffcde56547e
VRI: vcr-aware-journal.com/
cd6e83d7-3929-4452-9259-fffcde56547

Figure 3: Gaussian image inpainting. From left to right: original, degraded and recovered image. (Type/scan the URL, or click on the images to browse the computation.)

5. Discussion

5.1. VCR in the context of commonly encountered difficulties.

The issues raised by advocates of Reproducible Research are illustrated by a list of everyday situations, faced by any practitioner of the current workflow in computational science: “I experimented interactively with the values of the tuning parameters in my algorithm, and I can’t anymore tell you which parameters actually produced the figure that I included in my publication. I’m just not sure.” ; “Five years after publication, I have no idea where to find the code-data folder that actually produced a specific figure I published.” ; “My student has graduated and none of my new students can pick up from where she left off. I have the nice figures we published. The actual code-data are either missing or are incomprehensible.” ; “I have read this interesting paper but don’t believe the figures. I tried to implement the approach described in words but can’t create any of the figures myself. I can’t afford to spend the three days required to understand what’s going on within the code-data folder made available by the author.” ; “I would like to apply my own method to the original data used in this paper but have no way to obtain this data.” ; “I cannot referee this paper because I cannot assess the validity of the authors’ claims without looking at the details of the actual computation that took place.” ; “I suspect that the authors of this paper privately changed the original dataset before analyzing it.” ; “I suspect that the authors of this paper tweaked the tuning parameters manually to obtain publishable results.”.

We hope that this paper made it sufficiently clear that these issues no longer exist in a community practicing VCR. Risking a more general statement, we claim that in essence, all these situations can be traced to three fundamental drawbacks of the current workflow:

1. *Text-based publications are not enough.* The traditional core of scientific publication is text, graphics and mathematical formulas. These descriptions are inadequate to convey understanding of complex computations. As a consequence, research results that are produced on a computer are often not well understood, not trusted, or cannot be reproduced by other scientists based on such publications alone.
2. *Every result is detached from its creating process.* A computational result is created carrying no record of the process that created it. In the current workflow, the link between result and process is extremely fragile. Once it is broken, there is no telling what were the conditions under which a particular result was created. Attempts to document these conditions *retroactively* is a highly error-prone and untrusted process.

3. *Files are inadequate for scientific communication.* The de-facto standard for communication between computational scientists is exchange of files containing the datasets (when making data available) and the graphical results (when publishing an article). Handling multiple copies of locally stored files is a highly error-prone process and an inevitable source of mistrust, as files can be withheld and changed retroactively, either intentionally or by mistake.

The VCR discipline is specifically designed to eliminate these drawbacks. To the best of our knowledge, as common and acute as drawbacks 2 and 3 are, they have never been explicitly addressed.

5.2. Advantages of using VCR

Practicing the VCR discipline pays substantial long-term benefits to the full range of scientific stakeholders:

Journal reviewers can much more easily evaluate articles that are 100% based on VCR. They can easily authenticate and validate any result. They can carefully evaluate, and even interact with (in a sense that is beyond our current scope), every computational result in a submitted article, understanding how the data were prepared and where it came from, identifying specific parameters that were used in running the underlying computational processes, and so on.

Journals themselves have a new role as operators of the VCR repositories. The repositories will receive a continual flow of server requests from readers following up on the papers they read, and from their computer programs that are accessing data or computational elements published at the repository. Hence journals are able to provide a new type of service, and facilitate exchange of ideas through their publication venue.

Journal readers obtain significant value from articles that are 100% based on VCR. In both online and paper publications, every figure with a VRI is an entry point to the creating computation, through which they can browse, understand and sometimes re-execute it. In their own future research work, they can rigorously and unambiguously identify specific results in published work (eg. specific table elements in an article or specific coefficients of a fitted model in another article), and import those results automatically into their own computations. Instead of working with downloaded and locally stored data files, which can be mislabeled by them or questionable to others, or with snippets of text pasted from various anonymous sources and compiled into a file – a highly error-prone process – one deals solely with VRI's.

Journal authors and individual researchers set up a private VCR repository (similar to the public one run by the journal), which they use to archive their daily work and share it remotely with collaborators. The personal repository is an online, highly detailed, fault tolerant laboratory journal (in the sense this term is used in the experimental sciences), whose entries are made automatically by the computer rather than manually by the researcher. Every result they ever produced as a VCR is inspectable on their personal repository. In other words, an individual repository has all the benefits of the traditional lab journal – and magically, it is self-filling.

research groups set up a group-access-only VCR repository which improves interaction among group members, regardless of their physical location, and provides the group with enduring memory of research performed that does not evaporate when group members change. Every significant computational experiment, performed by the group, is archived in full on the group repository and can be located using the VRI's on internal group slides, technical reports, thesis, etc.

Scientific communities gain improved trust and improved knowledge accumulation. With VCR, publication in a journal unavoidably means publishing the computation - including original data, source code, tuning parameters, order of execution, input and output variables that every function has at runtime, external dependencies, and results produced - on the journal's repository. Sharing a code other than the one that actually ran, or parameters other than the ones actually used, or withholding data - *becomes physically impossible.* As a result, readers' trust in the publications increases, and vital information about the research work is preserved for future readers.

Acknowledgments

MG is supported by a William R. and Sara Hart Kimball Stanford Graduate Fellowship and would like to thank Balasubramanian Narasimhan, Alon Shalita and Omer Tamuz for their helpful suggestions.

References

[1] K. a. Baggerly, K. R. Coombes, Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology, The Annals of Applied Statistics 3 (4) (2009) 1309–1334. doi:10.1214/09-AOAS291.
URL http://projecteuclid.org/euclid.aoas/1267453942

[2] J. P. A. Ioannidis, Why most published research findings are false., PLoS medicine 2 (8) (2005) e124. doi:10.1371/journal.pmed.0020124.
URL http://www.ncbi.nlm.nih.gov/pubmed/16060722

[3] Editorial, Closing the Climategate, Nature 468 (7322) (2010) 345. doi:10.1038/328003c0.

[4] J. F. Claerbout, M. Karrenbach, Electronic documents give reproducible research a new meaning, in: Proceedings of the 62nd Annual International Meeting of the Society of Exploration Geophysics, 1992, pp. 601–604. doi:10.1190/1.1822162.
URL http://link.aip.org/link/SEGEAB/v11/i1/p601/s1\&Agg=doi

[5] D. L. Donoho, An invitation to reproducible computational research., Biostatistics (Oxford, England) 11 (3) (2010) 385–8. doi:10.1093/biostatistics/kxq028.
URL http://www.ncbi.nlm.nih.gov/pubmed/20538873

[6] D. L. Donoho, A. Maleki, I. U. Rahman, M. Shahram, V. Stodden, Reproducible Research in Computational Harmonic Analysis, Computing in Science & Engineering 11 (1) (2009) 8–18. doi:10.1109/MCSE.2009.15.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4720218

[7] S. Fomel, J. Claerbout, Guest Editors' Introduction: Reproducible Research, Computing in Science and Engineering (2009) 5–7.

[8] J. Buckheit, J. Buckheit, D. Donoho, Wavelab and reproducible research, Springer-Verlag, Berlin, 1995.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.6201

[9] R. Gentleman, D. Temple Lang, Statistical analyses and reproducible research, Journal of Computational and Graphical Statistics 16 (1) (2007) 1–23.
URL http://pubs.amstat.org/doi/abs/10.1198/106186007X178663

[10] Roundtable, Reproducible research, Computing in Science & Engineering 12 (5) (2010) 8–12.

[11] J. Freire, D. Koop, E. Santos, C. Silva, Provenance for Computational Tasks: A Survey, Computing in Science & Engineering 10 (3) (2008) 11–21. doi:10.1109/MCSE.2008.79.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4488060

[12] C. T. Silva, J. E. Tohline, Computational provenance, Computing in Science & Engineering 10 (3).

[13] B. Hanson, A. Sugden, B. Alberts, Making data maximally available., Science (New York, N.Y.) 331 (6018) (2011) 649. doi:10.1126/science.1203354.
URL http://www.ncbi.nlm.nih.gov/pubmed/21310971

[14] S. H. Koslow, Sharing primary data: a threat or asset to discovery?, Neuroscience 3 (April) (2002) 311–313.

[15] J. T. Overpeck, G. a. Meehl, S. Bony, D. R. Easterling, Climate Data Challenges in the 21st Century, Science 331 (6018) (2011) 700–702. doi:10.1126/science.1197869.
URL http://www.sciencemag.org/cgi/doi/10.1126/science.1197869

[16] R. E. Anderson, J. Gross, Mini-computers in a social science instructional context, Proceedings of the ACM annual conference on - ACM '72 (1972) 952–963doi:10.1145/800194.805883.
URL http://portal.acm.org/citation.cfm?doid=800194.805883