How to get reproducible computational experiments

Proposal for a thesis by Samuel Grayson

v

Feedback

Please deliver feedback (early and often) to:

Rubric

Intellectual merit, expected impact, feasibility

Takeaways

1. ∃ valuable but unexplored technical approaches to improve reproducibility
2. It is feasible to research those for my 12-month thesis.

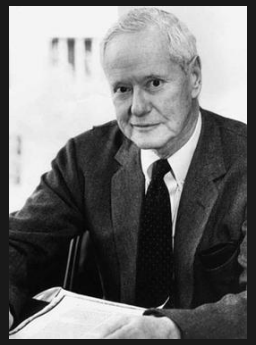
Contents

v

Motivation

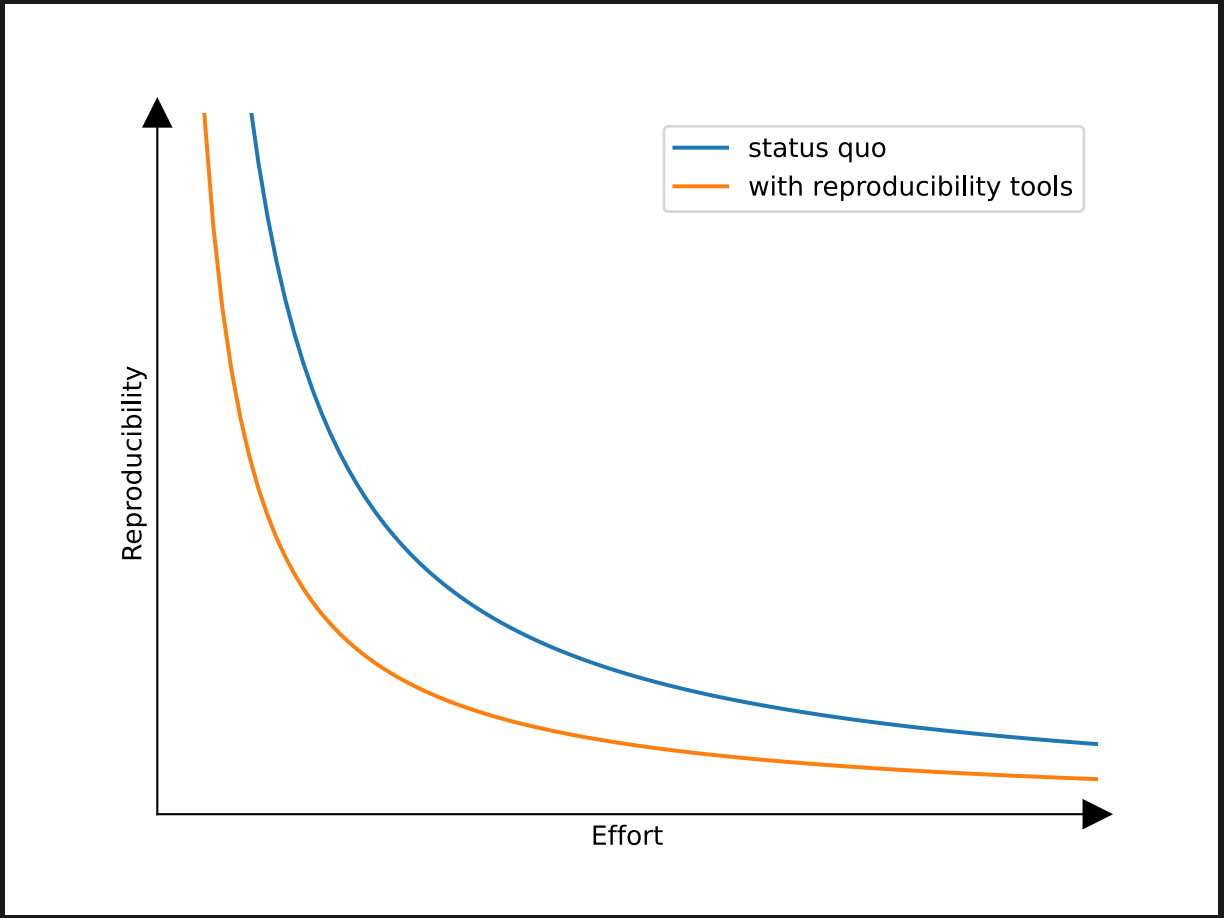
Why reproducibility

- Scientific claims should be scrutinizable
- Science builds off of other science
- Scientific research aims to impact practice
- Citation count (Hitchcock 2004)
- All of which require reproducibility



Robert K. Merton

Tradeoff between effort and reproducibility, but tech. improvement would help



Defining reproducibility

Reproducibility: The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials.

— (ACM Inc. staff 2020)

Related terms

	Repeatability	Reproducibility	Replicability
Experimental setup	Same	Same	Diff
Team	Same	Diff	Diff

$$f_h(x, c) = (y, c')$$

- f: run function
- x: user input (measurement procedure)
- y: user output (measurement)
- h: hardware and kernel (measurment system)
- c, c': configuration function: $\mathbb{C} \to \mathbb{A}$ (operating conditions)

Sufficient conditions for intended computation

$$c \in C_0 \implies f_h(x, c) = g(x)$$

As long as the configuration is set a particular way $\in C_0$, running the program $f_h(x, c)$ computes the expected result $g(x)$

Compositionality

$$f_h(x', f_h(x, c)_2)$$

Measurements

- Crash-freedom
- Bit-wise equivalence
- Statistics
- Inferential equivalence

Characterization of reproducibility in practice

Roadmap

- Prior work
- My work
- Conclusions

Is reproducibility a problem? How bad is it?

Has it improved over time or with tools?

Focus on the most frequent failure modes

Why not share source?

- G: Gomes et al. 2022, CP: Collberg, Proebsting, and Warren 2015
- Labor to maintain, distribute, or support (CP, G)
- Student who knows how left (CP)
- Unsure of process (G)
- Intellectual property issues (CP)
- Possibility of being scooped (G)
- Anxiety over public perception (G)

Characterizing reproducibility in prior works

- Collberg and Proebsting 2016
- Ferro and Kelly 2018
- Gunderson and Kjensmo 2018
- Krafczyk et al. 2019
- Neupane et al. 2019
- Trisovic et al. 2021
- Vandewalle et al. 2009
- Wang et al. 2021
- Winter et al. 2022

Conclusions from prior work

- Source availability began poorly, but improving (depends on the field) (V, G, C).
- Artifact badging itself seems to help (Wi, F)
- Src is buildable half of the time (C)
- Difficult to run notebooks and R, due to no environment (Wa, T)
- Results are consistent most of the time (K), although specific cases of irreproducibility are well-known (N)

The further down you go, the fewer samples are available:

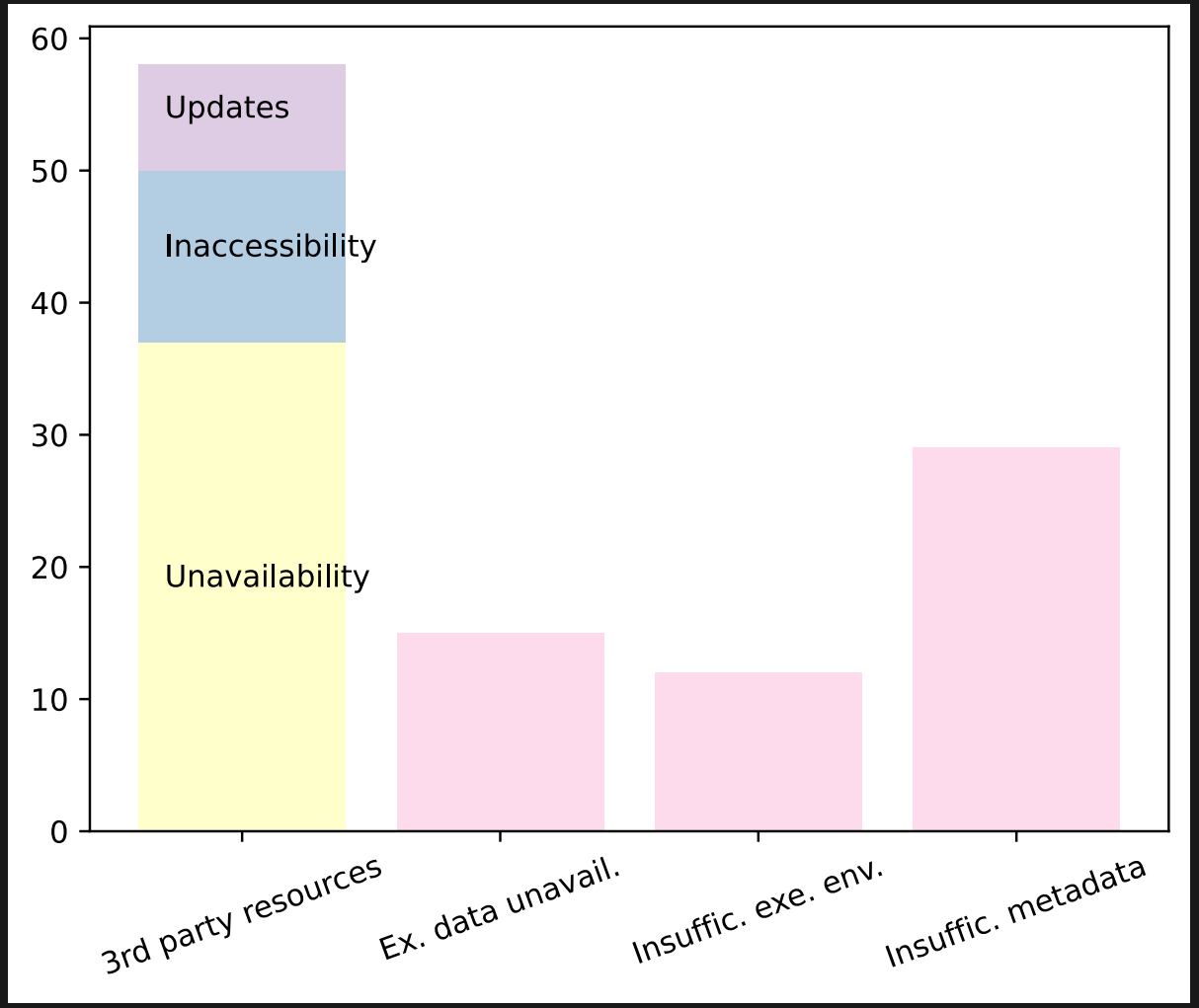
- locating source (V, GK, W)
- building source (CP)
- running code (Wa, T)
- re-examining results (K)

Workflows could offer structure that arbitrary source lacks

Reproducibility of workflows

- Workflow := language for digraph of coarse-grained tasks
- Taverna workflows on myExperiment (Zhao et al. 2012)
- 80% failed, mostly due to dependency on external resources
- What about modern workflows

Dist. of errors in Taverna workflows (Zhao et al.)



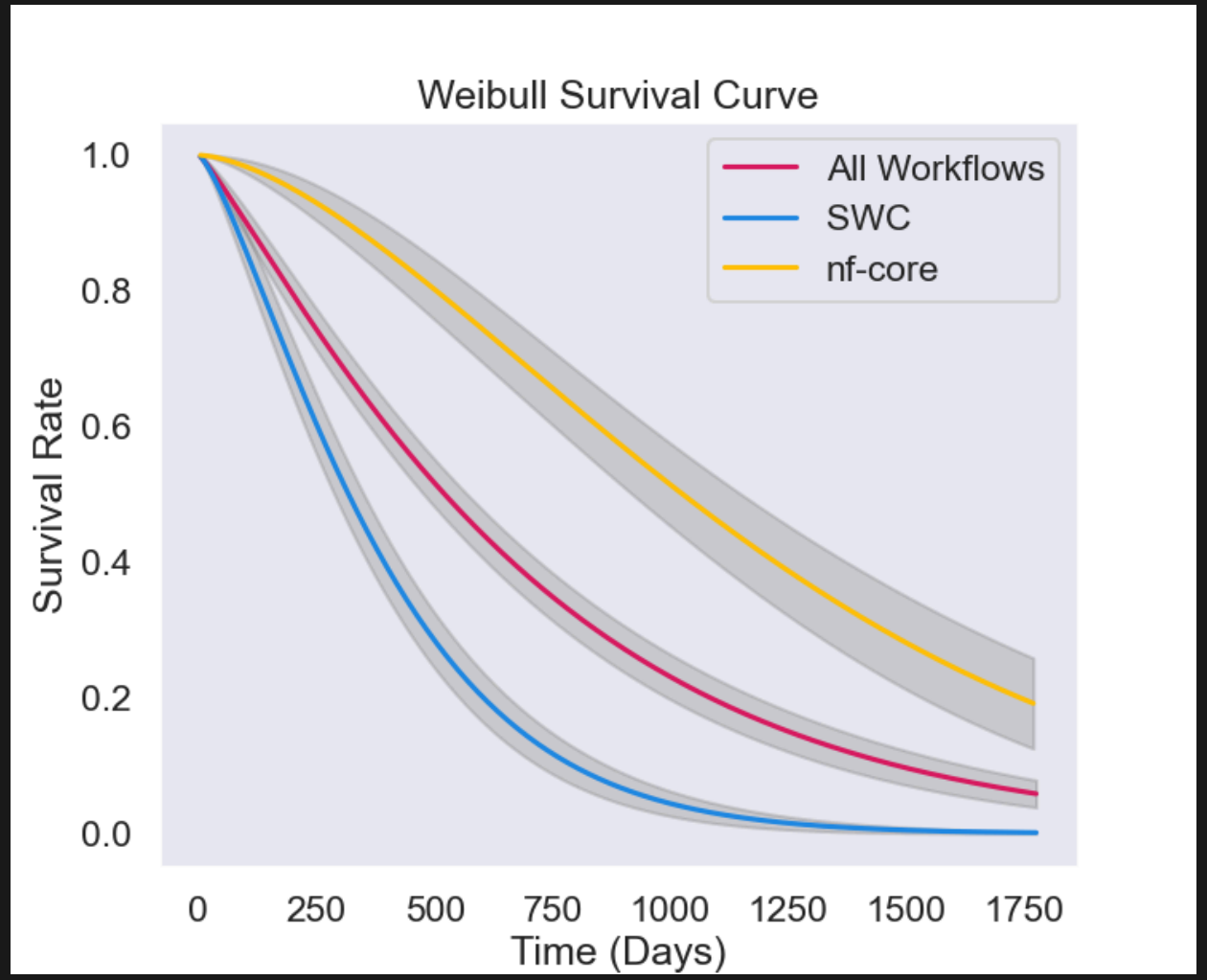
Characterizing reproducibility in selected workflows

- Workflows.community → workflow corpora
- Snakemake (2012 – present) and Nextflow (2013 – present)
- Each step can be containerized or Conda-sandboxed
- Should be better, but missing dependencies and data plagues new results

Distribution of error modes for Snakemake and Nextflow workflows in Grayson et al. 2023.

Kind of crash	All	SWC	nf-core
Missing input	32.2%	43.8%	16.7%
Conda environment unsolvable	10.8%	18.9%	0.0%
Unclassified reason	7.9%	12.0%	2.4%
Timeout reached	7.0%	5.7%	8.8%
Singularity error	6.0%	6.6%	5.2%
Other (workflow script)	5.7%	1.5%	11.2%
Other (containerized task)	1.2%	0.0%	2.8%
Network resource changed	0.7%	0.0%	1.6%
Missing dependency	0.5%	0.9%	0.0%
No crash	28.1%	10.5%	51.4%
Total	100%	100%	100%

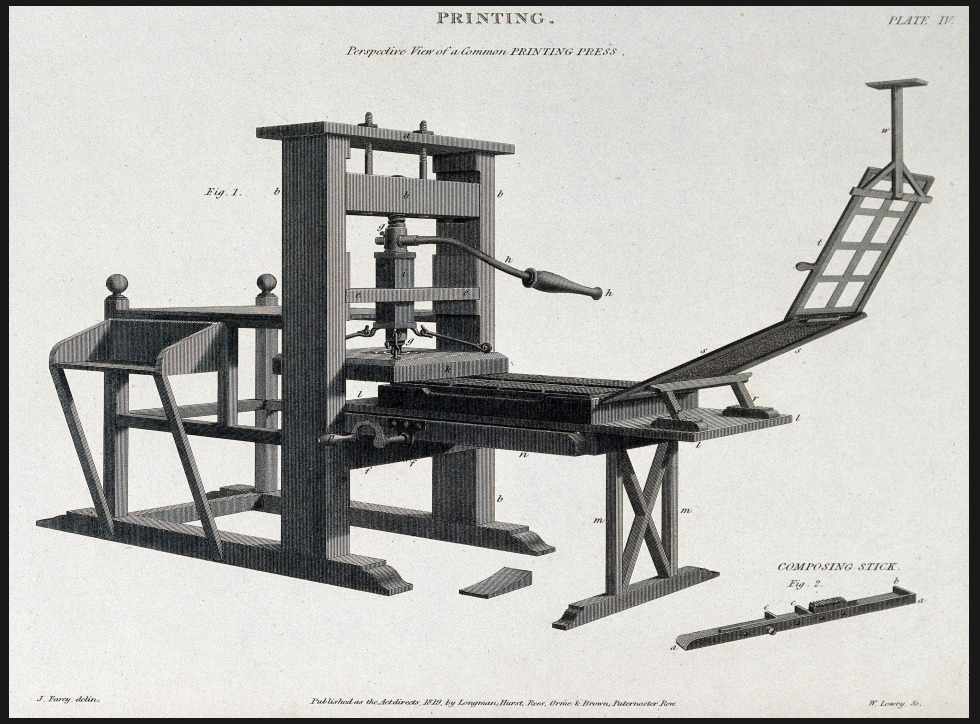
Survival analysis curves of workflows in Grayson et al. 2023.



Conclusions

- Primary reason: Labor required to package and explain
- Workflows and environments help, but not everything
- Average lifetime is 1 – 3 years

Technical approaches to reproducibility



Roadmap

- Prior approaches
- My work
- Conclusions

Prior approaches

- Virtualization, e.g., Docker
- Sys-level provenance, e.g., RR-debugger
- Notebooks, e.g., Jupyter
- Scientific clouds, e.g., WholeTale
- Workflow manager, e.g., Snakemake
- Package managers, e.g., Spack

Virtualization is not a solution

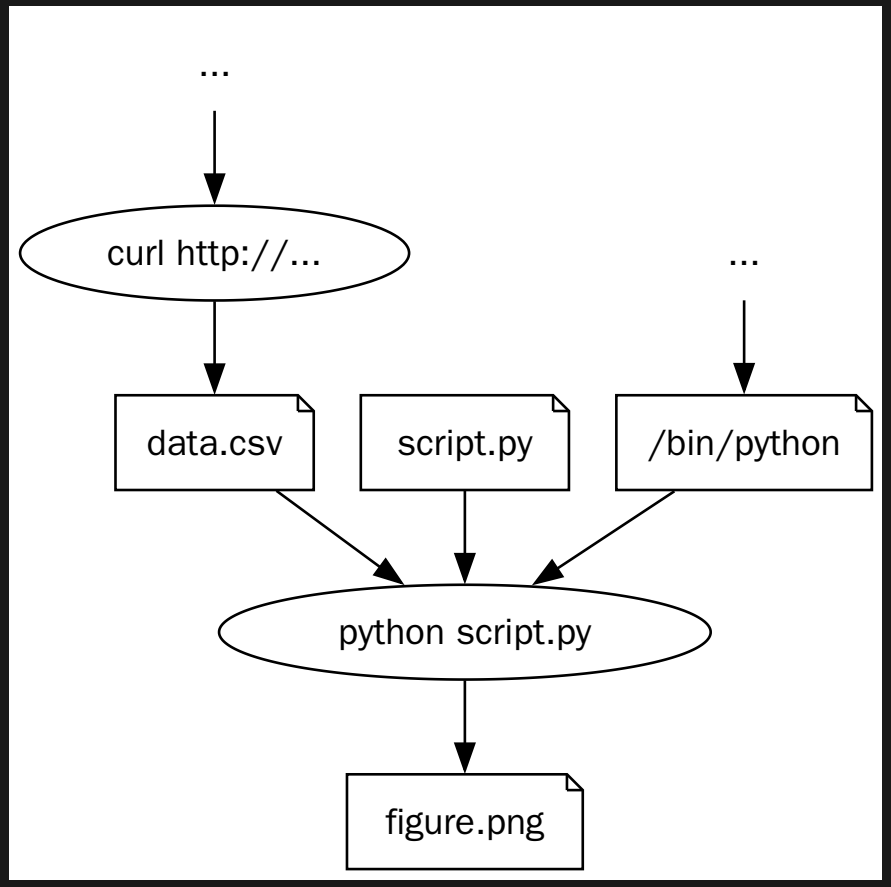
1. Distribute VM/container image
    - Too expensive to store many such images
    - DockerHub can delete after 6 months
2. Distribute VM/container build script
    - Is the base-image pinned?
    - Is the second line `apt-get update`?
    - This method can work, but it needs a reproible package manager

Provenance

1. Process by which a file was created
2. Inputs to that process
3. Provenance of those inputs

Provenance example

```
curl -o https://data.com
python script.py
```



Rapid review

Search queries: "Computational provenance" &
"System-level provenance"

v

Feasibility study

Linux sys-level prov tracers

Strategy	Modified kernel?	Privileges required?	Provenance tracer
Modify kernel	yes	yes	PASSv2, CamFlow, …
Built-in auditing	no	yes	SPADE, eBPF, …
User tracing	no	no	Sciunit, ReproZip, …

Performance

Benchmark	Native None	fsatrace Lib. interp.	CARE Ptrace	strace Ptrace	RR Ptrace	ReproZip Ptrace
BLAST	0	0	2	2	93	8
Tar Unarchive	0	4	44	114	195	149
Python import	0	5	85	84	150	346
VCS checkout	0	5	71	160	177	428
Compile w/Spack	0	-1	119	111	562	359
Postmark	0	2	231	650	259	1733
cp	0	37	641	380	232	5791
Others not shown	…	…	…	…	…	…
Geometric mean	**0**	**0**	**45**	**66**	**146**	**193**

Conclusion

1. System-level provenance tracing can address missing dependencies/data
2. Existing system-level provenance tracers require root or are too slow
3. Library interposition may be faster than traditional tracing

v

Proposed work

Roadmap

- Benchmark system-level provnenace tracer
- Complete large-scale reproduction of Collberg and Proebsting 2015

System-level provenance tracing

- Library interp. vs user-level tracing
- Evaluate robustness c.f. other methods, $\forall c \in \mathbb{C}, c \backslash c_{\text{fixed}}$
- Storage cost for daily use?
- Use interoperable standards

Reproduction of Collberg and Proebsting

- Availability or reproducibility of the original artifacts changed?
- Different results using the same procedure on newer data (same staleness)?
- Impact of artifact badge?
- Write machine-readable Dockerfiles and provenance traces (future re-analysis)

v

Conclusion

Takeaways

1. ∃ valuable but unexplored technical approaches to improve reproducibility
2. It is feasible to research those for my 12-month thesis.

Future work

- Use environment manipulation (from provenance) for automatic repair
- Impact of semantic version (from provenance) on reproducibility