Reproducibility is a Process, not an Achievement: The Replicability of IR Reproducibility Experiments

Jimmy Lin and Qian Zhang

David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada

This is the preprint of an accepted ECIR 2020 reproducibility paper.

Abstract. This paper espouses a view of reproducibility in the computational sciences as a *process* and not just a point-in-time “achievement”. As a concrete case study, we revisit the Open-Source IR Reproducibility Challenge from 2015 and attempt to replicate those experiments: four years later, are those computational artifacts still functional? Perhaps not surprisingly, we are not able to replicate most of the retrieval runs encapsulated by those artifacts in a modern computational environment. We outline the various idiosyncratic reasons why, distilled into a series of “lessons learned” to help form an emerging set of best practices for the long-term sustainability of reproducibility efforts.

Keywords: artifact evaluation · community benchmarks

1 Introduction

In the broad discussion on reproducibility in the computational sciences, there has not been, at least in our view, much discussion of the fact that reproducibility is an *ongoing process*, not just an achievement—a “checkbox” we “tick off” and then continue going about our business.

There are many reasons why reproducibility is a worthwhile goal, ranging from demonstrating good stewardship of public resources (who fund a large portion of research worldwide) to the fact that reproducibility is intrinsic to the scientific process itself. Reproducibility enhances the veracity of findings and supports the iterative process of knowledge accumulation through which researchers build on each other’s results.

The general discussion about reproducibility implicitly treats it as an achievement: a particular finding was reproduced successfully. And as a result, we (the scientific community) gain greater confidence in the veracity of the claims. This perspective is entirely appropriate in, for example, the physical or life sciences. Once we unravel the mystery of a particular physical phenomena, it is unlikely to change. Furthermore, reproduction efforts, for the most part, need to start “from scratch”, with a completely independent set of experiments. Although further studies refine our understanding over time (e.g., Newton's vs. Einstein's perspective on gravity), subsequent experiments are usually driven by new research questions, and cannot be considered reproduction efforts per se.

Reproducibility in the computational sciences, on the other hand, has many fundamentally different characteristics. A successful effort to reproduce a result typically yields a computational artifact that, ideally, should be executable by other researchers and remain functional over time. We explore exactly these desiderata and espouse the viewpoint that reproducibility in the computational sciences should not be viewed as an achievement (i.e., "we reproduced the technique of Yang et al. and confirm their findings") but rather a process.

Before proceeding, it is helpful to more precisely define our terminology. Here, we adopt recent ACM guidelines pertaining to artifact review and badging,[1] where reproducibility refers to artifacts created by an independent group to recreate a result, and replicability refers to using an existing artifact to recreate a result. Thus, we are concerned with the replicability of previous reproducibility (or replicability) experiments. That is, can we still rerun old computational artifacts to replicate their results?

As a case study, we attempt to replicate results from the Open-Source IR Reproducibility Challenge from 2015 [8] (henceforth, OSIRRC 2015). To quote: "the product is a repository that contains all code necessary to generate competitive *ad hoc* retrieval baselines, such that with a single script, anyone with a copy of the collection can reproduce *(sic)* the submitted runs." Four years later, can we still replicate those results? The answer, perhaps unsurprisingly, is *no* for most of the artifacts in a modern computational environment.

One might wonder, why would anyone want to run old code, and who cares? We present two compelling scenarios: First, findings in the computational sciences are always couched in some computational context. Take a simple example: algorithm A is faster than algorithm B, but the experiments were conducted on magnetic disks. With memory-resident data structures, does the finding still hold? Verification of results should be a continual process as computational contexts change: the shift from magnetic disks to SSDs, the advent of multi-core computing, the changing performance characteristics of memory vs. network, the evolving nature of on-premise, in-cloud, and edge execution, etc. Although we do not undertake such an examination in this paper due to limited space, being able to replicate results using old computational artifacts is a pre-requisite. Second, another compelling scenario is long-term software preservation [11], especially for artifacts that have historical value, in the same way that there is active interest in preserving old video games today. For example, the SMART system [14] might fall into this category.

The contribution of this work is a discussion as well as a specific case study examining the long-term sustainability of reproducibility efforts. We grapple with issues that are under-explored in the community—for example, our questions do not naturally fit into the Platform, Research goal, Implementation, Method, Actor, and Data (PRIMAD) model [6] that attempts to capture the multi-faceted

[1] `https://www.acm.org/publications/policies/artifact-review-badging`

aspects of reproducibility. Of course, platforms change over time, but this change is an inevitable consequence of the passage of time; in other words, "platform" is a dependent variable, not an independent variable, in our conception. PRIMAD implicitly assumes a static view of reproducibility, as opposed to the *process*-oriented view we advocate. More concretely, from the successes as well as failures in replicating OSIRRC 2015, we are able to extract a number of "lessons learned" that can be further distilled into best practices, especially to inform ongoing efforts such as the latest iteration of OSIRRC [3].

2 Replication Study

OSIRRC 2015 billed itself as providing scripts that allow any researcher to recreate, "out of the box", a number of baseline runs on the Gov2 web collection on topics 701–850 from the TREC Terabyte Tracks (2004–2006) [4]. Our experiments put these claims to the test. We cloned the git repository[2] and proceeded to run the `dotgov2.sh` script associated with each system after changing the location of the document collection in a common settings script. As designed, the single script should handle all aspects of producing an *ad hoc* experimental run from scratch, including building the inverted index, performing retrieval, and using `trec_eval` to generate effectiveness figures. In a few cases, the script worked exactly "as is", but more often than not, we encountered failures, which we then spent some time debugging (details below).

We ran experiments on two different servers, which we refer to as "old" and "new" for convenience. The "old server" was purchased in Spring 2017 and runs Ubuntu 16.04.2, with Oracle Java version 1.8.0_121 and `gcc` 5.4.0. The "new server" was purchased in Summer 2019 and runs Ubuntu 18.04.2, with Oracle Java version 11.0.2 and `gcc` 7.4.0.

The results of our replication study are summarized in Table 1; we refer the reader to the caption for details. In summary, we were able to replicate results from five of the seven systems on the old server, and for two we obtained *exactly* the same AP scores; the remaining systems saw minor score differences. On the new server, we were only able to replicate results from one of the systems. Details for each system are presented below:

ATIRE [15] and **JASS** [9] are closely related in that the latter depends on the former. On the old server, the `dotgov2.sh` script worked without modification for both systems, and for ATIRE we obtained exactly the same AP scores as in the OSIRRC 2015 paper. Unfortunately, ATIRE fails to compile on the new server, and thus we were unable to replicate results there (and JASS by extension). Both systems are implemented in C++, so the critical difference here is the version of `gcc` (5.4.0 vs. 7.4.0); the compilation error arises from namespace clashes and different definitions of `std::unordered_map` between C++11 and C++17.

Indri [12] results were replicable on the old server without any modifications to the `dotgov2.sh` script, but the AP scores differed slightly from those reported

[2] `https://github.com/lintool/IR-Reproducibility`

Table 1. Summary of our replicability experiments, reporting average precision (AP) across different topics for different system combinations. Bolded headings represent rows copied directly from the OSIRRC 2015 paper; (old) and (new) refer to replication attempts on our "old" and "new" servers, respectively. AP scores that differ from OSIRRC 2015 are enclosed in parentheses.

			Topics		
System	Model	Index	701–750	751–800	801–850
ATIRE	BM25	Count	0.2616	0.3106	0.2978
ATIRE (old)	BM25	Count	0.2616	0.3106	0.2978
ATIRE (new)	BM25	Count	— Failed: compile error —		
ATIRE	Quantized BM25	Count + Quantized	0.2603	0.3108	0.2974
ATIRE (old)	Quantized BM25	Count + Quantized	0.2603	0.3108	0.2974
ATIRE (new)	Quantized BM25	Count + Quantized	— Failed: compile error —		
Galago	QL	Count	0.2776	0.2937	0.2845
Galago (old)	QL	Count	— Failed: exception —		
Galago (new)	QL	Count	— Failed: exception —		
Galago	SDM	Positions	0.2726	0.2911	0.3161
Galago (old)	SDM	Positions	— Failed: exception —		
Galago (new)	SDM	Positions	— Failed: exception —		
Indri	QL	Positions	0.2597	0.3179	0.2830
Indri (old)	QL	Positions	(0.2746)	(0.3182)	(0.2893)
Indri (new)	QL	Positions	— Failed: segmentation fault —		
Indri	SDM	Positions	0.2621	0.3086	0.3165
Indri (old)	SDM	Positions	(0.2624)	(0.3079)	(0.3244)
Indri (new)	SDM	Positions	— Failed: segmentation fault —		
JASS	1B Postings	Count	0.2603	0.3109	0.2972
JASS (repl. old)	1B Postings	Count	0.2603	(0.3108)	0.2972
JASS (repl. new)	1B Postings	Count	— Failed: compile error —		
JASS	2.5M Postings	Count	0.2579	0.3053	0.2959
JASS (repl. old)	2.5M Postings	Count	0.2579	0.3053	0.2959
JASS (repl. old)	2.5M Postings	Count	— Failed: compile error —		
Lucene	BM25	Count	0.2684	0.3347	0.3050
Lucene (old)	BM25	Count	0.2684	(0.3346)	0.3050
Lucene (new)	BM25	Count	— Failed: exceptions —		
Lucene	BM25	Positions	0.2684	0.3347	0.3050
Lucene (old)	BM25	Positions	0.2684	(0.3346)	0.3050
Lucene (new)	BM25	Positions	— Failed: exceptions —		
MG4J	BM25	Count	0.2640	0.3336	0.2999
MG4J	Model B	Count	0.2469	0.3207	0.3003
MG4J	Model B+	Positions	0.2322	0.3179	0.3257
Terrier	BM25	Count	0.2432	0.3039	0.2614
Terrier (old)	BM25	Count	0.2432	0.3039	0.2614
Terrier (new)	BM25	Count	0.2432	0.3039	0.2614
Terrier	DPH	Count	0.2768	0.3311	0.2899
Terrier (old)	DPH	Count	0.2768	0.3311	0.2899
Terrier (new)	DPH	Count	0.2768	0.3311	0.2899
Terrier	DPH + Bo1 QE	Count (inc direct)	0.3037	0.3742	0.3480
Terrier	DPH + Prox SD	Positions	0.2750	0.3297	0.2897

in the OSIRRC 2015 paper. On the new server, Indri compiles successfully but immediately encounters a segmentation fault when starting to build the index.

Galago [2] retrieval runs aborted due to a thrown exception after the indexing process finished without any obvious signs of error; the same issue was encountered on both servers. The exception appears to come from TupleFlow, a custom MapReduce-like framework used by Galago, which never gained widespread adoption and thus likely suffers from robustness issues.

Lucene runs successfully without any modification to the `dotgov2.sh` script on the old server; we encountered two tiny differences in AP scores compared to the OSIRRC 2015 paper, we suspect due to tie-breaking effects [10]. Lucene, however, failed to run on the new server, throwing exceptions during indexing. The critical difference is the version of Java (8 vs. 11). In the Lucene version used in these experiments (5.2.1), a bug was noted in Oracle's JRE implementation of `MMapDirectory`, which is unable to properly close the underlying OS file handle. The workaround deployed at that time used an undocumented internal cleanup functionality, which no longer works on Java 11.

MG4J [1] failed for the simple reason that the `dotgov2.sh` script attempts to fetch a tarball that was no longer available at the specified URL.

Terrier [13] was the only system where the `dotgov2.sh` script worked without modification on both the old server and the new server. Furthermore, we obtained exactly the same AP scores as the OSIRRC 2015 paper. However, the paper references two retrieval models, "DPH + Bo1 QE" and "DPH + Prox SD", which do not appear to be included in the execution script. As a result, we were unable to replicate those two runs.

After encountering the initial errors with each of the systems, we did undertake efforts to debug the various issues. Ultimately, we did get two additional systems (ATIRE and Lucene) working on the new server. However, we consider it unreasonable to expect such debugging efforts from an envisioned user of the OSIRRC repository. Anything other than scripts "just working" should be consider a replicability failure, and a failure of OSIRRC to deliver on its promise.

3 Lessons Learned

For OSIRRC 2015, the passage of time has been ruthless in destroying the functionality of the computational artifacts. Although each artifact broke in its own idiosyncratic way, we do notice common themes. Given continued interest in reproducibility, most notably a new Docker-based iteration of OSIRRC in 2019 [3],[3] these "lessons learned" might form the basis of future best practices.

Where do these results come from? In many cases, the correspondence between figures in the paper and execution traces of the computational artifacts (even when successful) was not clear. As a simple example, Table 1 originally contained an "All" column, which, inexplicably, does not simply appear to be the average

[3] `https://osirrc.github.io/osirrc2019/`

of each set of topics. The execution trace of each `dotgov2.sh` script was different: some printed out AP scores directly to stdout; others didn't, and silently completed execution. In the latter cases, it was necessary to dig through system outputs to find the actual scores; the fact that each system used completely different naming conventions for output runs, evaluation results, and log files made this task non-trivial.

Due to space restrictions, we focused on replicating effectiveness results in this paper, but OSIRRC 2015 also evaluated index size and query latency. However, the repository offered few details on how to extract figures corresponding to those reported in the paper. As one example, it wasn't clear where each system stored its index: some were contained in directories, some were single files, and others were groups of files (and in many cases, not intuitively named).

The high-level lesson here is that researchers should strive to make the correspondence between execution traces and reported figures in papers as explicit as possible. This can be viewed as an endorsement of "executable papers" [7, 5]. Although this idea dates back decades (cf. literate programming), the popularity of notebooks (e.g., Jupyter) makes this much easier to realize today.

External dependencies. Six of the seven systems in OSIRRC 2015 began by pulling in an external dependency (namely, the search engine itself) over the internet; the only exception, Lucene, had jars directly checked into the repository. "Link rot" is a well-known phenomenon, and the replication effort for MG4J failed at the outset because the artifact was no longer available. To ensure the long-term availability of resources at stable URLs, we advocate the use of dedicated archiving services such as Zenodo (or other domain-specific data repositories), which also provide citeable DOIs.

Platform dependencies. As the saying goes, "change is the only constant", and computing platforms inevitably evolve, often in non-compatible ways: the two poignant examples here being JVM differences and compiler differences. Unlike external dependencies discussed above, it would be impractical to directly store and manage platform dependencies inside the OSIRRC 2015 repository, but here the use of Docker in OSIRRC 2019 represents an improvement. Docker allows platforms to be isolated in "layers" that can be composed to form images in a lightweight manner (compared to VMs); images themselves can be stored at stable locations (for example, in Docker Hub). Admittedly, this design does not guard against the obsolescence of Docker itself; despite the popularity of Docker today, one day it too will be superseded by new practices.

4 Conclusions

The 2019 edition of OSIRRC represents the next iterations of community efforts to advance the cause of reproducibility in information retrieval. It seems that many of the lessons discussed in the previous section have already been incorporated into its design. However, only time will tell if those efforts are successful: it would be interesting to try and rerun those experiments in 2023, the same distance we are from the initial OSIRRC efforts in 2015.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and a Postdoctoral Fellowship in Software Curation grant from the Council on Library and Information Resources (CLIR), made possible by funding from the Andrew W. Mellon Foundation.

References

1. Boldi, P., Vigna, S.: MG4J at TREC 2006. In: TREC (2006)
2. Cartright, M.A., Huston, S., Field, H.: Galago: A modular distributed processing and retrieval system. In: SIGIR 2012 Workshop on Open Source IR (2012)
3. Clancy, R., Ferro, N., Hauff, C., Lin, J., Sakai, T., Wu, Z.Z.: Overview of the 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). In: CEUR Workshop Proceedings Vol-2409. pp. 1–7. Paris, France (2019)
4. Clarke, C., Craswell, N., Soboroff, I.: Overview of the TREC 2004 terabyte track. In: TREC (2004)
5. Dittrich, J., Bender, P.: Janiform intra-document analytics for reproducible research. Proceedings of the VLDB Endowment **8**(12), 1972–1975 (2015)
6. Ferro, N., Fuhr, N., Järvelin, K., Kando, N., Lippold, M., Zobel, J.: Increasing reproducibility in IR: Findings from the Dagstuhl seminar on "reproducibility of data-oriented experiments in e-science". SIGIR Forum **50**(1), 68–82 (2016)
7. Gorp, P.V., Mazanek, S.: SHARE: A web portal for creating and sharing executable research papers. Procedia Computer Science **4**, 589–597 (2011)
8. Lin, J., Crane, M., Trotman, A., Callan, J., Chattopadhyaya, I., Foley, J., Ingersoll, G., Macdonald, C., Vigna, S.: Toward reproducible baselines: The open-source IR reproducibility challenge. pp. 408–420. Padua, Italy (2016)
9. Lin, J., Trotman, A.: Anytime ranking for impact-ordered indexes. In: ICTIR. pp. 301–304 (2015)
10. Lin, J., Yang, P.: The impact of score ties on repeatability in document ranking. In: SIGIR. pp. 1125–1128. Paris, France (2019)
11. Matthews, B., Shaon, A., Bicarregui, J., Jones, C.: A framework for software preservation. International Journal of Digital Curation **5**(1), 91–105 (2010)
12. Metzler, D., Croft, W.B.: Combining the language model and inference network approaches to retrieval. Information Processing & Management **40**(5), 735–750 (2004)
13. Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Lioma, C.: Terrier: A high performance and scalable information retrieval platform. In: SIGIR 2006 Workshop on Open Source IR (2006)
14. Salton, G.: The SMART Retrieval System—Experiments in Automatic Document Processing. Prentice-Hall, Englewood Cliffs, New Jersey (1971)
15. Trotman, A., Jia, X.F., Crane, M.: Towards an efficient and effective search engine. In: SIGIR 2012 Workshop on Open Source IR (2012)