Recomputation.org: Experience of Its First Year and Lessons Learned

Ian P. Gent
School of Computer Science
University of St Andrews
St Andrews, Fife
Scotland
Email: ian.gent@st-andrews.ac.uk

Lars Kotthoff
INSIGHT Centre for Data Analytics
Cork Constraint Computation Centre
University College Cork
Western Gateway Building
Cork, Ireland
Email: lars.kotthoff@insight-centre.org

***Abstract*—We founded recomputation.org about 18 months ago as we write. The site is intended to serve as a repository for computational experiments, embodied in virtual machines so that they can be recomputed at will by other researchers. We reflect in this paper on those aspects of recomputation.org that have worked well, those that have worked less well, and to what extent our views have changed on reproducibility in computational science.**

I. Recomputation: The concept, the name, and the manifesto

Reproducibility in Computer Science is in crisis, although nobody seems to have noticed yet. Actually, this is not nobody, as there are many people working in the area. But the mainstream of Computer Science has not realised that Computer Science – a science which should be more reproducible than most – is in danger of becoming less reproducible than most.

We have proposed that one part of the solution to the reproducibility problem is 'recomputation'. By recomputation we mean the act of embodying a computational experiment in a virtual machine so that it can be preserved and rerun at any point in the future. This is one very specific aspect of reproducibility, and far from the most important one. But it has two key advantages. First, once an experiment is made recomputable it can be rerun very easily. Second, a recomputable experiment can be preserved indefinitely.

The original idea for the name 'recomputation' would have been 'replication'. However, that would be too confusing in Computer Science since replication is a well established word in databases. We chose the name for two simple reasons "recomputation":

- the word "recomputation" is an existing word, with the right overtones and with no major meaning in Computer Science (although it has been used in various contexts) and
- the domain name recomputation.org was available.

In this paper we review the notion of recomputation and summarise the principles. We discuss a number of areas where our thinking on recomputation has changed over the last eighteen months, or where our thinking has become more nuanced by experience. We report on our experiences of giving two tutorials at conferences and soliciting and obtaining experiments for recomputation.org. Finally we discuss the major obstacles we face in making recomputation.org successful.

The recomputation manifesto puts forward the following points:

1. *Computational experiments should be recomputable for all time.*
2. *Recomputation of recomputable experiments should be very easy.*
3. *Tools and repositories can help recomputation become standard.*
4. *It should be easier to make experiments recomputable than not to.*
5. *The* only *way to ensure recomputability is to provide virtual machines.*
6. *Runtime performance is a secondary issue.*

For more discussion of these points as we understood them originally, see [1].

It has been noted that there are many words around this area [2]. Therefore there is some slight guilt about having introduced a new word or repurposed an existing word, to add confusion. However we hope that, if anything, we can reduce confusion. The recomputation manifesto is clear that the word is taken to mean the exact duplication of an existing experiment, including any flaws in the original.

II. Reflections on the Recomputation Manifesto

A. Recomputation: what is it good for?

Without doubt the recomputation manifesto laid out one goal for recomputation. This was the exact replication of previous experiments in the environment in which they were originally performed. This remains the focus of recomputation.org. Can we recompute an experiment in the way it has been done in the past?

This gives us a certain advantage in focussing on not a computational environment but a specific experiment. That is, we do not say the approach of embodying an experiment in a VM is original (see for example Titus Brown's experiment [3], [4]). However, we believe there are two key advantages. First,

computational experiments are the key drivers of empirical computational science. A paper making an advance will report on a specific experiment. This may be intended as an exemplar of others, but reviewers and readers will only have the specific experiment as evidence. This is the experiment that needs to be recomputed. Second, by wishing only to make specific experiments recomputable, we have greatly simplified the problems faced in providing a computational environment. It is not critical to have an extensive testing suite, because it is not necessary for anything not required for the experiment to work.

Those are the advantages of our focus on individual experiments. However, there is a potential downside. Our focus on individual experiments can seem a little sterile. We have an experiment, and then we keep it. It allows scientists to go back in time, but very often they do not want to.

We believe strongly that recomputation has major advantages beyond simply going back in time. We draw an analogy with source code control. In explaining the idea to a newcomer, one might explain that the key point is that one can go back in time. But there is so much more to it than that. Experienced users wonder how they ever lived without it. There are so many working styles it enables that are more productive than without it. These are the key advantages of source code control. In a similar way we see many advantages of recomputation.

Perhaps the most important advantage is that VMs can serve as either black boxes or as clear boxes. As black boxes, they serve to make experiments very easy to recompute. But for those wishing to delve deeper, they can be examined internally by logging in and looking around. Scientists can explore the machine, both statically (looking at files) and dynamically (trying variants of experiments). Ideally, an experiment would contain all relevant source code, but even if closed source, future researchers can see the exact environment a binary was run in.

The second – and very tightly linked – advantage is that a VM can be adapted for other purposes. Because it is a clear box, new scientists can see the great experiments of past scientists, or (if they are not great experiments) correct their mistakes. In either case, new experiments should be easier to construct and better than if each scientist had to build their own experiments. Computer scientists will have the incredible luxury of being able to work in each other's labs at a moment's notice and without invitation.

Third, there is a particular advantage of VMs for scientists who wish to make their codebases available. Having a VM in which an experiment is known to work might save great amounts of time for individual scientists or small research groups wishing to make their code available but without the resources to spend large amounts of time on flexible build systems and extensive documentation.

B. It was twenty years ago today

If you go to recomputation.org, the tagline you see is: "If we can compute your experiment now, anyone can recompute it 20 years from now." This is a key commitment, because there seems no limit to how long an experiment should be kept. Indeed, Computer Science is no longer a young science and 20 years is not such a long time in its history. We also have the advantage that in general terms, as time goes by old experiments get cheaper to keep and rerun because of Moore's law. However, two problems raise themselves if we are to keep experiments for twenty years.

The first is that it is vital not to lose data! A project founded on a twenty year promise would die immediately (or at best become a well-deserved laughing stock) if it lost old experiments. In this first period we have not focussed on this enough. We have kept our VMs on single machines or backed up to another machine in the same physical rack, meaning potential disaster in the case of a single fire. We have recently improved this so that now we are immune to a single fire, but have yet to develop serious long term preservation plans. To do that we are looking at services such as Amazon Glacier, and Arkivum, the latter especially being heavily focussed on guaranteeing long term preservation of data. Data preservation thus becomes a financial instead of a technical problem. To indicate finances, Arkivum's prices start at £1 per GB per year, but academic discounts are available as are discounts for volume in both amount of data and period of storage. Many machines we store are less than 1GB, so indeed 20 year storage for such a machine can be guaranteed (to within reasonable human limitations) for less than the cost of a single meal at the conference the experiment was described at.

The second issue is more problematic. This is to ensure that we can still recompute experiments in 20 years that are deposited now. There are many issue around this. First, a new version of virtual machine software could mean an experiment stops working. We can keep old versions of virtualisation software around, but as time passes that software itself may stop running on then-generation hardware. At this point the issue becomes problematic. Two main options seem available. First is to keep old hardware which can run the old virtualisation software to run the even older experiment. The second is to emulate an older machine on a newer machine: this could be either the machine the original experiment was run on, or the machine that the virtualisation software was run on. Our suspicion is that we may need both approaches. Emulation imposes heavy overheads, but is effective [5]. Very many, perhaps most, old machines can be emulated. This is particularly clear in the case of games, where almost all old games consoles are now emulated to an incredible degree of fidelity. Many games depend on undocumented features of the architecture discovered by games designers, yet they can be run in emulators. For example the Olive Executable Archive not only does this for games and other software, but allows streaming over the internet [6]. This can be hoped for in future for current machines. Indeed, if there were a major architectural change, it is likely that major effort would be made by third parties to provide emulators, because of the huge investment in legacy software: this happened for example when Apple changed Macs from PowerPC to Intel chips.

In summary, is reasonable to think that we can run 20 year

old experiments but this will need ongoing and significant care and attention, and this is something that we have not yet given to this issue.

C. Recomputation ≠ Exact Reproduction of Results

Possibly the most controversial point in the recomputation manifesto is that runtime results are secondary. The reason this is controversial is that often, from the point of view of the scientist, almost the only point of an experiment is runtime, to see for example if a new method is faster (that is, assuming that correctness is already known.) Yet, in a VM, one might get very different results – even opposite – from an original set of experiments on bare silicon ran on a different architecture to the machine doing the emulation.

We have discussed this at length with many people. We wish to lobby against one – typically unstated – assumption underlying this debate. This is that there is such a thing as the true result, e.g. runtime speedup, that can be captured in some way. Instead, we argue that in reality we must expect to get different results every time we run an experiment, even when we recompute using the identical machine each time. We do not see recomputation as the same thing as exact reproduction of results. Indeed, this goes beyond varying measures such as runtime, and can affect results obtained such as the predicted weather in one day's time.

There is an analogy with the famous movie *Groundhog Day*. In this movie the character experiences the same day, February 2, over and over again. Everything is the same except the character's response to the situation. Indeed the phrase "Groundhog Day" has become synonymous with events repeating as well as its original meaning of a certain day of the year. The analogy is clear. The same day is experienced over and over again, in the same environment. But we get subtly different results each time. This is akin to a recomputation: the same calculation performed in the same environment over and over again, but with different results. In recomputation, this could be because of many factors. Naturally it could be due to different CPU speeds or random number generators, but it could be due to almost arbitrarily minor points. For example, floating point addition is not associative, so if we were simulating more than one core, the result of major computations can be sensitive to, for example, the temperature outside a data center. A slight increase in temperature outside might increase the air conditioning, cooling down one core slightly, leading partials to arrive in a different order, and a different final sum to arise in an unstable system.[1]

When we run experiments repeatedly, we can expect to see some aspects remain the same, and others to vary. Often the main result will be the same but runtimes vary. But this is to be accepted – even celebrated – and absorbed into our culture. There is a huge difference between saying 'Algorithm A ran 1.2 times faster than Algorithm B on one machine once', and saying 'Algorithm A ran a median of 1.2 times faster but varied from 1.05 to 1.40 times faster over 47 runs in different environments'. The latter is far more informative. And recomputation provides researchers the tools to collect more data, so need not be a sterile act. Rerunning experiments may well be adding to the store of human knowledge, not just checking that our existing knowledge is correct.

At the end of *Groundhog Day*, after millions of repetitions (one estimate is that the main character experiences 10,000 years' worth of days), the character has learnt something incredibly powerful about himself. Perhaps there will be experiments which – after having been recomputed millions of times – lead to powerful results that could not have been found from a small number of runs.

D. Comparison With Other Approaches

The use of virtual machines for archiving computational experiments is neither original with us, nor the only possible approach. There are numerous other approaches which have been or can be used to package up experiments. These fall into three categories. First, there are packaging systems which have been developed not as tools for experiments, but as tools to package software for distribution. Examples would be Docker [7], Nix [8], and CDE [9]. These are all attractive because they allow software to be distributed in ways which allow execution without having the exact same environment as the original machine. A second set of tools are those developed with a particular programming language or environment in mind. Particularly good examples would be iPython notebooks [10] and Active Papers [11]. The uniformity of language means that portability is wide, but does limit generality to experiments using these languages. There is also a problem with versions of languages changing between the original and the reproducing environment. A final type of tool is one which is not limited to certain languages but is explicitly designed to host experiments. The prime example is reprozip [12], which is explicitly a tool for packaging experiments.

There are significant attractions of these methods. One advantage these approaches tend to have over recomputation is that the object to be distributed is usually much smaller than a whole VM. They also have a general but often minor disadvantages. This is that, to a greater or lesser extent, there is still a dependency on the original software base. Sometimes this is very broad, e.g. CDE only requires the same major kernel version (e.g. 2.4) of Linux, making it a fairly minor point.

We see recomputation as working very well with these methods. The very fact that these methods can be used to make experiments replicable relatively easily means that we can embody the method in a VM and then use it to create a VM containing the experiment. This serves both to allow use to combine other techniques with recomputation, but also to underwrite the other techniques. That is, if say a reprozip experiment is recomputable now, it does not matter if some later change to Linux invalidates the reprozip experiment, as it can still be run in our VM.

There does remain a significant drawback of these methods. This is that the experimenter has to use them. This sounds a

[1]We are grateful to Steve Linton for suggesting the data center example.

triviality, but we wish to minimise the demands we make on experimenters, to maximise the chance that they will be able to work with us. For example, an experimenter using reprozip has to use reprozip for all parts of the experiment, and not forget to use it for one part. In contrast, if an experimenter can give us a virtual machine in which a kludgy set of experiments is held together with string with no principled methodology, if we can recompute it now we can recompute it in 20 years.

III. RECOMPUTATION.ORG

As we mentioned, the name recomputation arose because the domain name was available. So we bought it, and since then it has pointed at machines in the University of St Andrews. We always expected to use Virtualisation tools such as VirtualBox: a particular advantage of this tool is being free and mainly open source. We were, however, delighted to discover the existence of Vagrant, an additional tool which also makes it particularly easy to use VirtualBox, as well as working with other virtualisation environments such as VMWare. Vagrant provides 'boxes', which are single files containing virtualised machines plus metadata. With these two tools in place we were able to move ahead with recomputation.org.

The two key principles of recomputation.org have been

- to deal in individual scientific experiments, or a number of experiments combined in a single VM, and that
- it should be very easy indeed to recompute one of our experiments.

From these two principles, plus the use of Vagrant, we established the desideratum that when a user fires up one of VMs, the experiment in that machine should run automatically. This is easily achieved in vagrant through the use of 'provisioning scripts'. As an example, we provided an experiment for a simple puzzle involving placing queens on the board [13]. We have created a video which shows the experiment being rerun at [14]. The point of this is that a user who has VirtualBox and Vagrant can rerun this experiment with no knowledge of the technology underlying it, or any additional installation of software or tools. The experiment runs automatically and deposits the results in the user's space on the host machine, so there is no need even to log in to the guest machine. When the experiment finishes the user can look at the results and, if they wish, log in to the guest machine to investigate the setup in greater detail.

After eighteen months, we have about twenty experiments stored at recomputation.org. Most of these have come from two conferences where we worked with the programme chairs, inviting authors to make their experiments recomputable. We discuss these two conferences in detail below.

A. Tutorials

We ran two tutorials at international AI conferences to get people interested in recomputation. In both cases, we did not just present our ideas on recomputation, but also invited the authors of accepted papers at the respective conference to make their experiments recomputable. The benefits of this approach are twofold. On one hand, authors got hands-on experience with making experiments recomputable, and on the other hand we became aware of issues with recomputability in practice.

The approach we took to making experiments recomputable was to create virtual machines (VMs) with Virtualbox[2] and Vagrant[3]. The VMs contain everything required to run (and in some cases, analyse) the experiments done in the corresponding paper published at the conference.

The first conference we considered was the 19th International Conference on Principles and Practice of Constraint Programming in Uppsala, Sweden, in September 2013 (CP 2013). This is the main conference for the constraint programming community and had about 150 participants. The second conference was the 21st European Conference on Artificial Intelligence in Prague, Czech Republic, in August 2014 (ECAI 2014). ECAI is one of the three big AI conferences and had about 400 participants.

The main differences between those two conferences for us were their sizes and scope. Both authors of this paper are very familiar with constraint programming. The advantage for CP 2013 was that the contents of the papers and experiments was understandable for us. This was not the case for all experiments we made recomputable for ECAI 2014, as many more areas of AI, in which we have no background, were within the scope.

This, together with the larger scale of ECAI, necessitated a different approach to making experiments recomputable. While for CP, we asked the authors of accepted papers to package their experiments in a way that they thought they could be reproduced and send them to us. We then created the virtual machines to encapsulate the experiment, transferred the files the authors sent to us, and installed all the necessary software. This was a labour-intensive manual process that only worked because the number of experiments was relatively low and we were familiar with many of the software tools used in the experiments.

For ECAI, we took an entirely different approach. We provided the authors with access to a web interface that allowed them to create, control, and package virtual machines themselves. Instead of requiring busy academics to install virtual machine software on their own machines, everything was handled in the browser. We assumed basic familiarity with Linux and provided basic instructions on how to use the interface.

For CP 2013, we had 11 expressions of interest from authors, which resulted in 6 recomputable experiments by the time of the tutorial. For ECAI 2014, we received 14 expressions of interest and had 6 recomputable experiments when we gave the tutorial. In both cases, the tutorials themselves were well-attended and received, in particular at ECAI where we presented to a packed room with only standing room left.

[2]https://www.virtualbox.org/
[3]https://www.vagrantup.com/

B. Lessons learned

We became aware of many issues while working with paper authors in the run-up to the tutorials. Many of these raised limits of recomputability. Examples of these include experiments that involve entities that are inherently not recomputable, such as human subjects. This is an issue in human-computer interaction research, where humans evaluate for example user interfaces. While the evaluation itself cannot be made recomputable with virtual machines, we can at least make the analysis of the results recomputable.

Another limitation is posed by the resources some experiments require. In one case, an author told us that running the experiments he had done for the paper required about 1.5 years of CPU time. In another case, the experiments relied on a graphics card with a GPU from a specific vendor. While such experiments can still be packaged in a virtual machine in a way that they are recomputable, there is no guarantee that each individual is necessarily able to recompute them due to a lack of resources.

We also encountered legal issues when making some of the experiments recomputable. In one case, the authors compared their approach to a proprietary commercial tool. While the company provides free licences to academics, we were obviously not allowed to distribute this tool in a virtual machine. We opted for a pragmatic solution of this issue and gave the user the option of providing the tool themselves in a directory that would be cross-mounted in the virtual machine such that the experimental setup would pick it up and use if present, else ignore the part of the experiments that required it.

Finally, there were a number of issues that provide excellent motivation for our approach of using virtual machines. One set of experiments required the installation of a theorem-proving infrastructure that is difficult to install. Indeed, it does not compile out of the box on a standard Linux installation. This is obviously a major obstacle for potential users of the software, especially when trying to recompute the experiments of other researchers. Providing a VM with everything pre-installed makes this much easier, and indeed the authors of this particular experiment expressed their interest in having a VM available not only for recomputation, but also as a basis for future experiments.

In general, a major obstacle for recomputation in our experience is the effort involved in making experiments recomputable and convincing researchers to invest this effort. The numbers for the tutorials show that although there was quite a lot of initial interest in our initiative, this translated to actual recomputable experiments only in a fraction of the cases. For CP 2013, where we put in most of the work ourselves, the "conversion ratio" was higher than for ECAI 2014, where we asked the authors to do most of the work.

In order for recomputation to become more widespread, we need to continue working on making it as easy as possible for experimenters to make everything they are doing recomputable. In addition, we need to work on making recomputability a standard practice in the Computer Science community.

IV. Key Obstacles To Recomputation

There are many issues which could face recomputation.org, and our campaign for recomputation in general. However, we single one out in particular. This is the ease of making experiments recomputable. To us, this is critical. We have made it easy to recompute an experiment (courtesy of Vagrant and VirtualBox), but it is not (yet) easy to take an experiment you have just finished and make it recomputable.

We have taken two approaches, as outlined above. The first, from CP 2013, was to get experimenters to send us raw files and for us to convert them manually. However this approach is highly non-scalable. The other approach, from ECAI 2014, was to give researchers access to VMs in which they could install required software and build their experiments. This is more scalable (though still heavy in computational resources at our end) since it uses much less of our time. However, it imposes significant learning curves on experimenters. We try to make these bearable with documentation and providing one-to-one help, but this is still a significant problem.

We feel that this is the key obstacle to recomputation at the minute because it is also a point in the manifesto: It should be easier to make experiments recomputable than not. At the moment, we do not know how to achieve this. There are excellent approaches such as reprozip [12] which move towards this, but they have a fundamental disadvantage: one has to use the tool throughout building the experiment. We want to reach people who have just finished an experiment and only now want to make it recomputable: rerunning the experiment in reprozip can be approximately the same level of work to them as building the VM.

Just now we do not know how to achieve this goal. The best we can offer is to chip away at it. This means incrementally making it easier to contribute experiments. Also it means that we should provide people as many routes to giving us experiments as possible. For example, if people are comfortable with Docker, we should be able to take an experiment packaged as a Docker application and run an experiment in a VM prepared to accept it. If that works we have now got the experiment in a VM, without the user having to move outside their comfort zone. The same would go for reprozip, ipython notebooks, and any other tool that people might use to build and run their experiments.

We believe firmly that there is not a "one true way" to build and run experiments so we have to be flexible. But we hope, either individually or in partnership with tools like reprozip and Docker, to make it easy to build and run experiments and make them recomputable. As we do this, we believe that people will see the benefits of running experiments in a recomputable way, just as they now seize the benefits of source code control and insist that their collaborators use them too.

V. Conclusions

As well as ease of use, there are other important directions we wish to progress in. For example, we feel that it would be productive to work with conferences and journals at the

submission stage, and indeed before that. By making recomputation available to prospective authors and reviewers, authors would not have to make experiments recomputable after the fact, but could do this from the start. Reviewers could look in detail in experiments *in silico*, and published papers would be more likely to have recomputable experiments.

We have enjoyed the first eighteen months of recomputation.org, the interest it has garnered, and all the very interesting discussions we have had on it with many people.

Most importantly, our key ambition has not changed. We wish to play our part in changing the way Computer Science is done.

ACKNOWLEDGEMENTS

We especially want to thank three groups. First, we thank very much all those who have deposited experiments at recomputation.org. Their names can be found by exploring recomputation.org. Second, those who have done a lot of the work behind recomputation.org, namely Lakshitha de Silva, John McDermott, and Alexander Konovalov. Third, we want to thank those who have provided financial or other assistance in various ways. This includes in particular the University of St Andrews, University College Cork, the Software Sustainability Institute, Microsoft Azure, an EPSRC Impact Acceleration award, and the conferences CP 2013, ECAI 2014 and the summer school EMCSR 2014.

Naturally we want to thank very much the very many scientists we have discussed recomputation with, but they are too numerous to list individually.

REFERENCES

[1] I. P. Gent, "The recomputation manifesto," Apr. 2013.

[2] C. Goble, "Results may vary: reproducibility, open science, and all that jazz," http://www.slideshare.net/carolegoble/ismb2013-keynotecleangoble, 2013, iSMB/ECCB Keynote.

[3] C. T. Brown, "Our approach to replication in computational science," http://ivory.idyll.org/blog/replication-i.html, Apr. 2012.

[4] C. T. Brown, A. Howe, Q. Zhang, A. B. Pyrkosz, and T. H. Brom, "A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data," 2012.

[5] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 41–41. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247401

[6] "Olive executable archive," https://olivearchive.org/.

[7] "Docker: Build, ship and run any app, anywhere," https://www.docker.com/.

[8] "Nix: the purely functional package manager," https://nixos.org/nix.

[9] P. J. Guo, "CDE: A tool for creating portable experimental software packages," *Computing in Science and Engineering*, vol. 14, no. 4, pp. 32–35, 2012.

[10] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org

[11] K. Hinsen, "Activepapers - computational science made reproducible and publishable," http://www.activepaper.org.

[12] F. S. Chirigati, D. Shasha, and J. Freire, "Reprozip: Using provenance to support computational reproducibility," in *5th Workshop on the Theory and Practice of Provenance, TaPP'13, Lombard, IL, USA, April 2-3, 2013*, A. Meliou and V. Tannen, Eds. USENIX Association, 2013.

[13] I. P. Gent, "Our first experiment: a chess puzzle," http://www.recomputation.org/blog/2013/07/01/our-first-experiment-a-chess-puzzle, 2013.

[14] ——, "Recomputation chess puzzle," https://www.youtube.com/watch?v=vlV1whY9VJc, 2014.