Unit (and other) testing of stochastic code

Simon Dobson

Complex and Adaptive Systems Research Group
School of Computer Science, University of St Andrews

`mailto:simon.dobson@st-andrews.ac.uk`
`https://simondobson.org`

University of St Andrews
FOUNDED 1413

Introduction

We've recently been working on epidemic modelling

- Simulation of stochastic processes on networks
- Large scale (10^5 nodes), lots of repetitions at different points in a parameter space

Developing and maintaining a codebase for stochastic processes has raised some interesting questions about how to engineer such systems

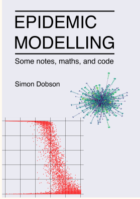
This talk

- How we optimised, what went wrong, questions that arise about software engineering for stochastic codes

EPIDEMIC MODELLING ON ONE SLIDE

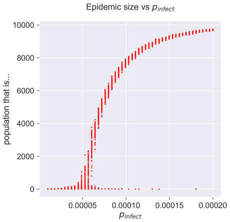
"Compartmented" disease models [1]

- *e.g.*, **S**usceptible-**I**nfected-**R**emoved
- Maintain sets of **SI** edges and **I** nodes
- Draw random element and change edge states, node compartments
- Sequential, several million operations



Scale

- Simulate for different disease parameters
- Exact results are stochastic, but follow distributions that are known for common processes (and not for others)



[1] S. Dobson. *Epidemic modelling – Some notes, maths, and code*. Independent Publishing Network, 2020. ISBN 978-183853-565-0. URL https://simoninireland.github.io/introduction-to-epidemics/

The optimisation

Core operation is drawing a random element from a set

- Python's inbuilt sets don't support this
- ⇒ re-code as balanced binary trees
- "Book" solution involves lots of random numbers
- ⇒ developed an optimisation that reduced this significantly
- Massively faster – at the cost of introducing a slightly biased choice, some elements slightly less likely to be drawn

Question: Is this optimisation safe?

Let's do some experimental science

Distribution of SIR and other processes are known

- We were already aggressive about testing, with a full CI infrastructure in place
- Analytic prediction of the location of the phase transition
- $\Rightarrow$ run a set of sample experiments, compare empirical to theoretical distribution
- χ^2 goodness-of-fit test (or other statistical magic)

Therefore although we know that there *is* bias, it isn't being observed by the disease process

EVERYTHING ALWAYS WORKS, UNTIL IT DOESN'T

Several months later, combine disease process with addition-deletion process

- Dynamic population of nodes, changing population of edges
- Addition-deletion has a known final degree distribution [2]
- …and our implementation doesn't follow it

Much debugging later

- The addition-deletion process *does* observe the bias
- (Still not entirely sure why…something to do with the time nodes are resident in the sets)

[2] C. Moore, G. Ghoshal, and M. Newman. Exact solutions for models of evolving networks with addition and deletion of nodes. *Physical Review E*, 74, September 2006. URL `doi://10.1103/PhysRevE.74.036121`

Local solutions: better unit tests

Stochastic code needs specific kinds of test

- A result isn't right or wrong *on it's own*, and therefore isn't (on its own) a suitable unit test

Each test samples the distribution of possible results

- Take samples, compare to what's expected
- (We've written a library to do this, obviously)

Challenges

- You need to run lots of samples, which may be individually expensive
- (Do you *really* want to *need* a compute cluster for testing?
- May not know the distribution you should expect

Where are the stochastic elements?

In our case we *knew* we had stochastic effects

- We were looking at the shapes of distributions (although not in the place that affected them)

What happens when you *don't* know?

- Race conditions can be very subtle
- (Does your OS thread scheduler affect your results?)
- More importantly, these effects can come from the *interactions* between components rather than from the components themselves

When is stochastic code stochastic?

The interactions are themselves stochastic

- Some processes observe bias, some observe variance
- …and some don't

The risks

- The composition of two correct components may not be correct – a massively larger surface area for testing
- By definition less likely to observe low-probability events
- We have a weak understanding of these effects
- How does a stochastic operation map the distributions of its inputs to those of its outputs?

Specifying test suites

Was there something we could have done differently?

- How to design suites that catch these effects?
- You can only deliberately test something you can observe (and know you want to)

More-than-unit tests

- We took to reproducing the results of known and "classic" papers, in whose results we had confidence
- Do our simulations get the same results?
- This then threw up some major questions about reproducubility and the adequacy of the scientific paper as a communication tool…

Conclusion

We need to look at this further

- Changes the way we think about testing
- Changes the testing infrastructure
- Makes automation/devops/CI even more important (but potentially more resource-intensive)

How much stochastic code is out there?

- We don't know
- Comes up very obviously in simulation, which is perhaps something we need to teach more of
- Important as an application area, but also illuminates issues of general software engineering interest

References

S. Dobson. *Epidemic modelling – Some notes, maths, and code*. Independent Publishing Network, 2020. ISBN 978-183853-565-0. URL `https://simoninireland.github.io/introduction-to-epidemics/`.

C. Moore, G. Ghoshal, and M. Newman. Exact solutions for models of evolving networks with addition and deletion of nodes. *Physical Review E*, 74, September 2006. URL `doi://10.1103/PhysRevE.74.036121`.