DebugNS: Novelty Search for Finding Bugs in Simulators

David Griffin
Computer Science
University of York
York, UK
david.griffin@york.ac.uk

Susan Stepney
Computer Science
University of York
York, UK
susan.stepney@york.ac.uk

Ian Vidamour
Computer Science
University of Sheffield
Sheffield, UK
i.vidamour@sheffield.ac.uk

***Abstract*—Novelty search is used to find a range of novel behaviours in a system. Software bugs are behaviours that are a) unexpected and b) incorrect. As the intersection between "novel" and "unexpected" is non-empty, here we overview how novelty search can be employed to find bugs in simulation software. We give an example of this approach applied to the RingSim simulator.**

***Index Terms*—novelty search, debugging, simulation**

I. Introduction

Novelty Search [1] is a common technique used to find a range novel behaviours of a system under study. While it may not be able to find optimal solutions to a given problem, novelty search subverts issues commonly found in optimising search algorithms, such as plateaus and local optima. This makes Novelty Search (NS) an appropriate method for exploring a landscape of behaviours; even if the behaviours are similar in nature, Novelty Search frameworks such as CHARC [2] are capable of exploring the space of possible behaviours.

In many cases, NS is applied to simulations, since it is typically easier to expose parameters to Novelty Search in simulation than in a physical experiment. However, simulators are software, and therefore prone to bugs in implementation. These bugs can range from coding errors, running software outside of its designed parameters, or unintended interaction of systems. The effect of bugs can vary, but at least some will manifest as the simulator behaving differently from what was expected. The NS approach can also be used to expose simulation bugs.

When NS is applied to a simulation for experimental purposes, then development effort will have be placed into connecting the simulator to NS. As this effort is expended, it makes sense also to utilise NS to characterise how the simulator behaves. The information on the behaviour landscape can then be tested against the developer's understanding of the system, to help identify potential bugs.

The CHARC framework [2] uses NS to characterise the behaviour of reservoir computers, using a range of computational measures [3]. DebugNS is adapted from CHARC; instead of using standard reservoir metrics, the user supplies measures that characterise what the system under analysis does. While typical testing approaches might use binary tests, such as comparing against expected output, DebugNS investigates the behaviours of the system for unexpected properties.

The authors would like to acknowledge funding from the MARCH project, EPSRC grant numbers EP/V006029/1 and EP/V006339/1.

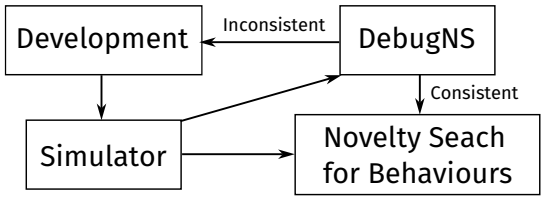

Fig. 1. DebugNS Workflow

II. DebugNS

DebugNS implements NS using the microbial genetic algorithm [4], although this could be replaced by a variety of other search techniques. This core is paired with a number of user defined benchmarks to define a behaviour space. In simulators, these measures will typically be derived from the properties of the system being simulated, at various levels. NS is then used to find input vectors for the simulator that result in maximally unique behaviours, as characterised by the measures, exploring the user-defined behaviour space. Figure 1 gives an overview of how a developer might use DebugNS.

Once the landscape of behaviours has been explored, it can be analysed to determine the likelihood of bugs. Simplistically this could be identifying regions of the landscape known to correspond to incorrect results; the simplest measure for DebugNS could be "does the program crash", which is clearly not correct behaviour; analysis determined parameter ranges where crashing occurs. Some measures might be known to be mutually exclusive, which would paint regions of the behaviour space as invalid.

More sophisticated analysis of the landscape can take into account the distribution of results. For example, one might have an intuition on the number of possible "classes" of results, with each cluster on the landscape corresponding to a class. If the number of clusters is either too large or too small when compared to the user's understanding of the system, this indicates either a flaw in the user's intuition or a bug in the simulator.

979-8-3503-1232-4/23/$31.00 ©2023 IEEE
DOI 10.1109/GI59320.2023.00012

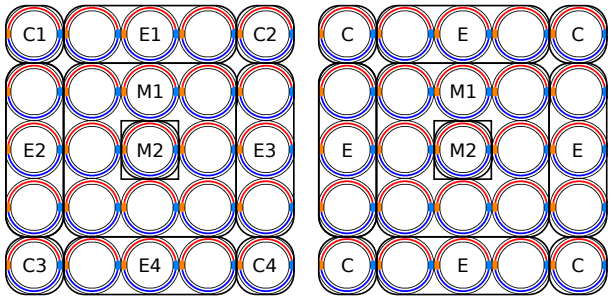

Fig. 2. Illustration of ring behaviour classes identified by DebugNS before (left) and after (right) debugging. Note that after debugging, behaviour classes are rotational- and mirror-symmetric, in-line with hypotheses.

III. Case Study: RingSim

RingSim [5] is a phenomenological-level simulator that models the movement and interactions of domain-wall phenomena driven by a rotating magnetic field in a square grid of nano-scale magnetic rings. DebugNS has been used to identify inconsistencies in the phenomenological model used by RingSim, with a particular focus on how the assumptions made by RingSim may not hold as it is generalised away from its initial design parameters.

In use, DebugNS controls the initial state of RingSim and the input sequence. Measures include the behaviour of the array as a whole, and also each of ring in the array. As RingSim is a simulation of a stochastic process, each experiment is repeated multiple times to find the mean and standard deviation of each measure. Once data is gathered, the k-means clustering [6] is used to assign each ring to a class of rings with similar behaviour; the number of clusters is defined to be the value that produces the tightest clusters. This approach suffices in this application, as only a small set of values have to be checked; for more complex scenarios, other approaches such as Principal Components Analysis [7] may be more appropriate.

Due to the phenomena RingSim simulates, the behaviour of a ring is influenced by its position in the array. However, from previous experimentation, the expectation is that only the local neighbourhood influences any individual ring on the timescale of input that DebugNS uses, resulting in only a relatively small number of classes of ring behaviours. Further, intuition on the system suggests that rotational and mirror symmetry should be observed in ring behaviours on a square grid. This allows the construction of the following tests after clustering:

- The number of ring behaviour classes is substantially smaller than the number of rings.
- Rings in rotational- or mirror-symmetric positions are in the same class.

DebugNS identified behaviours that violated the above tests, which were flagged to be reviewed as an "unexpected behaviour". By doing so, multiple issues were found when trying to generalise RingSim:

- Some calculations assume the magnetic field moves by rotating through the center of each ring, which could result in Domain Walls passing through each other.
- Some calculations assume a clockwise rotating magnetic field always means clockwise rotating Domain Walls (which, rarely, is not true).
- Due to RingSim sequentially updating each ring from top-left to bottom-right or the array, behaviour of top and left edges is slightly different to bottom and right edges (figure 2).

While DebugNS is able to find these kind of inconsistencies within RingSim, it is not able to find all potential issues. One issue not found was, when improvements made to the simulation, the value of one constant needed to be changed to ensure optimal behaviour. As the simulation was still consistent with a sub-optimal value, DebugNS did not identify this as a problem.

IV. Conclusions and Future Work

We have introduced DebugNS, a simple framework to take advantage of existing Novelty Search integrations to find bugs in simulation software. A case study demonstrates that DebugNS is able to find inconsistencies within a simulator, although unable to find bugs that do not cause inconsistent behaviour. We suggest that NS may be an appropriate way to debug (find unexpected behaviours) in a wide range of simulations.

We are extending DebugNS to allow a comparison between two systems, by exploring a landscape of differences between the systems. This has a range of potential applications. Where a trusted oracle is available, this could be used to compare the oracle to a simulator under development, for example, when developing a simulator from experimental observations. It should also be possible to use this method to compare two separate implementations, for example, two different simulations, or different physical implementations, and determine where they differ. This could be used to help determine the reproducibility of an experimental setup.

References

[1] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *ALife XI, Boston, MA, USA*. MIT Press, 2008, pp. 329–336.

[2] M. Dale, J. F. Miller, S. Stepney, and M. Trefzer, "A substrate-independent framework to characterise reservoir computers," *Proceedings of the Royal Society A*, vol. 475, no. 2226, 2019.

[3] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, 2007, pp. 471–482.

[4] I. Harvey, "The Microbial Genetic Algorithm," in *ECAL 2009, Budapest, Hungary*, ser. LNCS, vol. 5777. Springer, 2011, pp. 126–133.

[5] R. W. Dawidek, T. J. Hayward, I. T. Vidamour, T. J. Broomhall, G. Venkat, M. A. Mamoori, A. Mullen, S. J. Kyle, P. W. Fry, N.-J. Steinke *et al.*, "Dynamically driven emergence in a nanomagnetic system," *Advanced Functional Materials*, vol. 31, no. 15, p. 2008389, 2021.

[6] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[7] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.