Open Challenges in Genetic Improvement for Emergent Software Systems

Penny Faulkner Rainford[1, 2] and Barry Porter[1]

[1]Lancaster University
[2]p.s.rainford@lancaster.ac.uk

***Abstract*—Genetic improvement for emergent software systems faces unique challenges due to its deployment in highly dynamic environments. In this paper, we discuss four of those challenges along with our initial plans for new research.**

I. Introduction

Modern software systems are both highly complex, and also deployed into highly dynamic environments; together this makes it very difficult to quickly adjust a system's implementation to its current deployment conditions with small samples of the new environment. A recent example is the Zoom hosting servers which went from handling 10 to 300 million daily meetings between December 2019 and April 2020 [1]; such services also experience significant hourly fluctuations in demand both over time and geographic distribution.

Emergent software systems aim to mitigate these issues by autonomously and continuously discovering the ideal behaviour for a target system from among a large pool of small fragments of encapsulated logic (basic functions of 200 lines of code or less), and learning which of those compositions best suit the current deployment conditions at runtime to constantly adapt to the environment [2]. These systems rely on the existence of *variation* in their pool of building blocks to learn better compositions of behaviour – such as different sorting algorithms, cache replacement algorithms, or load balancing policies. While most systems require human engineers to generate this pool of variation, an initial study by McGowan *et al.* indicated that genetic improvement (GI) could automate the generation of environment-tailored variation with the example of a hash table [3]. Based on this study, we examine the wider set of challenges in genetic improvement for emergent systems and some of the most promising research directions to explore.

We begin by briefly reflecting on the key challenges in general GI, then examine the more specific challenges to emergent systems in how we navigate to different areas of a fitness landscape over time to solve changing problems.

In general, GI algorithms search a landscape of possible code that is functionally infinite in size and known to be extremely rugged [4]. The main challenge in these algorithms lies in the design of the fitness function [5, 6], which defines the phenotype (functional or non-functional properties of the software) and assesses the performance of our code; in particular, changes in the genotype (the code itself) have an unpredictable effect on the phenotype fitness function, which can make the genetic search process different to control.

When applying GI to emergent software systems we have a moving target in the deployment conditions of the system; in this case the code that we use to implement sorting, cache replacement, or load balancing can become not just sub-optimal but fundamentally the wrong approach. We need to transition from tuning existing code within these implementations to moving between wholly different forms of implementation. We need to do this quickly, with a limited view of the new environment providing minimal training data.

In the remainder of this paper we present four challenges for GI, and particularly for emergent software systems, with our initial plans for new research. These are (a) search space coverage with limited genetic source material; (b) generalisation techniques; (c) navigating between different implementation forms; and (d) leveraging human knowledge and creativity.

II. Challenges

a) Searching the space: Our base implementation for GI is always limited by the genetic material we can use; emergent systems consider encapsulated code of 100-200 lines, which exacerbates this challenge. While we can add new material with horizontal gene transfer [7], which has proven effective with evolved self replicators in dynamic environments [8], the question is where we get the material from.

Three main options exist here: open-source repositories such as GitHub; random genetic material; or a curated selection of material. GitHub is a great source of code but is also very large, posing search issues for *appropriate* material to insert. Random genetic material is limited by the random code we set up and may be less useful. Curated material from past success at injecting material, by comparison, may be a good way to better use either of these resources by storing material that has been useful in the past for faster access to good material.

A systematic study of these alternatives in the context of emergent systems would be highly informative to future GI research, and is one of our early objectives.

b) Generalisation: While optimisation to a given target is important, genetic improvement algorithms typically aim towards a fixed environment – for example optimising a particular component towards a specific set of inputs. Rather than using GI to derive a hash function for all words in the English language, we would instead use a smaller subset of commonly-seen inputs from a particular environment [3]. However, this presents a potential dichotomy between spe-

cialising on a smaller set of inputs or generalising on a much broader set – a problem analogous to over-fitting in wider machine learning and a significant problem in all evolutionary systems. In emergent software systems, which need to switch between different implementations, an extension of this challenge (beyond smaller training data sets) is on whether it is better to have many variants available for highly specialised roles, which expands the runtime search space for the system, or to aim instead for a few variants for major classes of environment that the system is likely to encounter.

To avoid extreme over-fitting in GI, we could examine the effects of using multiple training sets, randomly sampling from a larger training set [9], or periodically using a different sets either in blocks or as single generation events [10]. We could also examine the shape of data we use for our GI fitness function: for instance, to improve a hash function, instead of using a specific set of inputs observed in the running system, we may instead want to bias our training set to keys with a given length or character entropy to match a target language like English. In other words, can we shape our training data to better represent *classes* of data for which we want to derive a good implementation, rather than specific examples of a class?

A study of how we could better select or generate our GI training data, and the effect of this on over fitting when combined with methods for changing training data between generations, is another of our early research objectives.

c) Navigating between distant optima: In some cases, the optimisation of code to a given environment may require a larger leap. For instance, a bubble sort is a good solution for sorting a small amount of data but if the amount of data increases by a hundred fold then a more optimal sort algorithm might be shell sort. The difference between these two algorithms is huge in terms of their code, with large areas of the intermediate space having low or zero fitness [4]. It is almost impossible for classic genetic improvement methods to traverse this kind of search space – akin to evolving a shark into a dolphin; this inherently limits the degree to which an emergent system can optimise to its deployment environment.

We could approach this problem in different ways: by creating a system that specifically encourages large leaps across the search space [11], by searching in multiple areas at the same time [12], or by changing our fitness function to encourage movement towards handling the change.

These all have different challenges. Making large leaps is highly random and requires a large population to find the correct area. Searching multiple areas at once spreads resources thinner and requires some idea of areas to be searched. Trying to tailor a fitness function to change the space to encourage movement towards a different type of algorithm is a further challenge in generating a suitable fitness function.

d) Human Guidance: Human guidance already exists in GI systems through design choices: we choose the genetic material in the system, the possible mutations, the fitness function, the training data, and a stopping point.

Each of the above forms of human guidance apply to the initial creation of the GI system. Further forms of human input may be useful, however, *during* the GI process to boost its search power: humans could identify new search areas for distant optima, could add specific genetic material for injection to a curated set, update training data to correspond better to expected input, or even change the fitness function to better fit the priorities of the system and guide it to a desired outcome.

More ambitious still, human input could provide outlines for new kinds of algorithmic (rather than code) structures for the system to explore, or suggest points at which functional blocks become too big and need to be separated into sub-blocks. It is our long term goal to embrace human involvement in emergent software synthesis via GI, to gain the benefits of creativity and insight that machine learning algorithms still lack.

Summary In this paper we have examined some of the most pressing open challenges in GI for emergent systems, along with initial studies to begin exploring these challenges. We aim to initially focus on search space coverage and generalisation, then examine the more difficult topics of utilising human guidance and navigating between distant optima.

REFERENCES

[1] "Zoom revenue and usage statistics (2020)," https://www.businessofapps.com/data/zoom-statistics/, Apr. 2020, accessed: 2021-1-13.

[2] B. Porter and R. R. Filho, "Losing control: The case for emergent software systems using autonomous assembly, perception, and learning," in *IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems*, 2016, pp. 40–49.

[3] C. McGowan, A. Wild, and B. Porter, "Experiments in genetic divergence for emergent systems," in *4th International Workshop on Genetic Improvement*, ser. GI '18. Association for Computing Machinery, Jun. 2018, pp. 9–16.

[4] E. Pitzer and M. Affenzeller, "A comprehensive survey on fitness landscape analysis," in *Recent Advances in Intelligent Engineering Systems*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer, 2012, vol. 378, pp. 161–191.

[5] M. Harman and J. Clark, "Metrics are fitness functions too," in *10th International Symposium on Software Metrics, 2004. Proceedings.* Ieee, 2004, pp. 58–69.

[6] M. Harman and J. Petke, "Gi4gi: Improving genetic improvement fitness functions," in *Proceedings of the Companion of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 793–794.

[7] S. M. Soucy, J. Huang, and J. P. Gogarten, "Horizontal gene transfer: building the web of life," *Nat. Rev. Genet.*, vol. 16, no. 8, pp. 472–482, Aug. 2015.

[8] M. J. Wiser, R. Canino-Koning, and C. Ofria, "Horizontal gene transfer leads to increased task acquisition and genomic modularity in digital organisms," *Artificial Life Conference Proceedings*, vol. 31, pp. 243–244, Jul. 2019.

[9] I. Gonçalves, S. Silva, J. B. Melo, and J. M. B. Carreiras, "Random sampling technique for overfitting control in genetic programming," in *Genetic Programming*. Springer Berlin Heidelberg, 2012, pp. 218–229.

[10] I. Gonçalves and S. Silva, "Balancing learning and overfitting in genetic programming with interleaved sampling of training data," in *Genetic Programming*. Springer Berlin Heidelberg, 2013, pp. 73–84.

[11] Y. Fan and S. A. Sisson, "Reversible jump MCMC," *Handbook of Markov Chain Monte Carlo*, pp. 67–92, 2011.

[12] S. Chib and S. Ramamurthy, "Tailored randomized block MCMC methods with application to DSGE models," *J. Econom.*, vol. 155, no. 1, pp. 19–38, Mar. 2010.