The gem5 C++ glibc Heap Fitness Landscape

William B. Langdon
Department of Computer Science,
University College London,
Gower Street, London, UK
w.langdon@cs.ucl.ac.uk

Bobby R. Bruce
Department of Computer Science,
University of California, Davis,
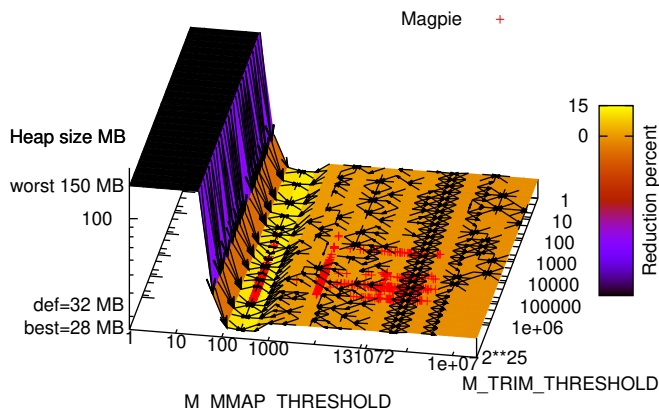Davis, California, USA
bbruce@ucdavis.edu

Fig. 1. Search landscape for the size of gem5's heap. GNU glibc has 7 parameters, two plotted. Setting mmap infeasibly small increases the heap size (red-blue-black), but for gem5 values 256–512 give a reduction ≈11% (bright yellow) relative to the default 131 072 (orange). Of the 7 dimensions, one (mmap) is dominant and it has a huge sweet spot. Note log scales.

Abstract—**Using the language independent genetic improvement tool MAGPIE (Machine Automated General Performance Improvement via Evolution of software) and logarithmic sampling, we measure the parameter fitness landscape when optimising the GNU glibc heap management of a million line C++ application, gem5. The malloc_info landscape is far smoother than is commonly assumed and savings of 11% with no loss of runtime speed are readily obtained by both Magpie and CMA-ES.**

Index Terms—**SBSE, computer program tuning, parameter search landscape, memory reduction.**

I. INTRODUCTION

It is sometimes assumed that the fitness landscapes [1]–[4], [5, Chapter 2] associated with software engineering problems are vast and very difficult. Indeed authors often like to boast that they have solved a particularly horrendous case. Here we take the opposite approach and show this need not be the case and that sometimes the search based approach [6] to a software engineering problem faces a landscape which is smooth, effectively unimodal and contains many acceptable solutions[1] (shown in yellow in Figure 1).

---

[1]In our example of reducing memory usage by at least 10% for a program containing more than a million lines of C++, the fitness landscape contains ≈ 400 201 439 265 161 216 ≈ 4 10^{17} solutions, i.e. not a trivial fraction of the whole search space.

The next section gives the background to: recent investigations of fitness landscapes in software engineering particularly for genetic improvement (Section II-A), gem5 (our million+ line C++ example, Section II-B), optimising gem5's heap memory consumption (Section II-C) and the size of the heap parameter's search space (Section II-D). The following sections describe reducing gem5's memory consumption by 11% using Magpie (Section III) and CMA-ES (Section IV), before we conclude that the difficulty of software engineering search spaces may sometimes be overplayed and sometimes they can be easy (Section V).

II. BACKGROUND

Genetic Improvement [7]–[11], has been applied to many aspects of software, including machine code [12]–[15], byte code [16], [17], assembler [18], LLVM intermediate code [19], [20], functional building blocks [21], quantum gates [22], even test suites [23], network protocols [24], story generation [25], dataflow programming languages [26], chemical reaction networks [27], SQL [28], mining specifications from code [29], reducing JavaScript load time [30] and security [31], [32]. It is often used for enhancing non-functional aspects of programs [33] such as reducing energy consumption [34], [35], network bandwidth [36], memory [37], [38] and run time [8]. GI has also been demonstrated with parallel programming [39]–[42] and even transplanting functionality from one program to another [43]–[45]. GI has also been proposed for exploring novel hardware designs before manufacture [46] and search [6] has been suggested to optimise software testing [47]. GI is increasingly being used with advanced AI techniques such as large language models [48], [49]. However GI tools such as GIN [50], [51] and Magpie [52] are usually applied to the human readable program source code itself. Although there have been limited target specific GI approaches which simultaneously optimised parameters and source code [41], [53], it appears that at present Magpie is unique in being the only general purpose GI tool which can tune both parameters and source code, e.g. [54]. Whereas CMA-ES [55] (Section IV) is the state of the art evolutionary algorithm for tuning continuous parameters.

A. Genetic Improvement Fitness Landscapes

As Petke et al. [56] point out (apart from their own work on open source Java programs) there are almost no systematic studies of the search landscape when modifying non-trivial programs. For example, we considered the Triangle Program benchmark [57]–[61], whilst Renzullo et al. investigate neutral networks in the Unix look utility [62] and the popular Defects4J automated repair benchmark [63].

Information theoretic approaches [64] have considered data flows within software [65] and concluded, due to entropy loss leading to failed disruption propagation [66], programs are much more robust than is commonly assumed [67]. Entropy loss means disruption of all sorts (e.g. bugs [68], mutations, even cosmic ray induced transients) can fail to reach the program's outputs and so have no impact, particularly in deeply nested code [69], [70]. Silent errors and mutation which do not change the output, typically do not change fitness. This leads to flat, plateau ridden, software fitness landscapes with many corresponding neutral moves.

As mentioned in the previous section, mostly automatic software improvement has considered the programs themselves, however many aspects of software are configurable and contain parameters which can be tuned [71] or can be deliberately exposed for optimisation [37]. For example, object-orientated code relies on dynamic memory structures under the control of the heap manager. In Section II-C we will describe in the context of our example program, gem5, the seven tunable GNU C/C++ malloc heap parameters, and how they can be automatically optimised to reduce memory consumption without rewriting source code.

B. gem5 and prior work on gem5

gem5 [72], [73] is the state of the art discrete time simulation tool for computer chips and hardware. It comprises a total of 1.34 million lines of code (mostly C++)[2]. It supports Intel X86, ARM, RISC V, MIPS, Alpha, Spark and Power instruction sets, as well as various memory chips and cache architectures. It is open source and has been under active community led development and maintenance by a diverse team for more than 13 years. (Umeike et al. [75] and Dakhama et al. [76] previously used gem5 as a benchmark.) Although it supports the LLVM compiler, gem5 is mostly used with the GNU g++ compiler and so uses the GNU C/C++ glibc runtime library.

C. glibc Heap and gem5

To show a software engineering example where the search landscape is much smoother than is commonly assumed (Figure 1) we will show tuning glibc's heap management for gem5.

The GNU C and C++ compilers share a common runtime library. In particular the C malloc family of dynamic memory management functions are also used by the C++ new and delete operators. Thus we can use the GNU C malloc_info function to report details of gem5's heap, particularly its

size. In common with other general purpose heap managers, glibc's imposes both runtime and memory overheads. (In the context of gem5, except for severely mistuned parameters i.e. timeouts, such as those shown in Figure 5, we saw little variation in runtime heap overhead.) Extra memory is used within the heap both for internal pointers to manage the heap and padding both: to control heap fragmentation and to ensure individual memory allocations are suitably aligned for the computer's hardware (we run gem5 on Intel's 64 bit X86 architecture). malloc_info and malloc_usable_size give details about actual memory on the heap. The memory actually allocated is typically larger than the application asked for (e.g. via new). glibc's malloc typically allocates small memory requests from its own heap and much larger request by directly asking the operating system for memory via mmap. The following sections describe some aspects of Linux' data memory management and alternative measurement tools.

1) gem5 Linux statm reports memory 4K pages: Under the Linux operating system the actual memory given to the program is available in the pseudo file /proc/self/statm (see man proc). statm reports the current number of pages allocated to a Linux process. These include both total virtual pages and number of pages actually in memory. Naturally active memory use depends not only on gem5 but also upon other processes in the computer and so is noisy. Nevertheless gem5 gave somewhat consistent results on our networked Centos 7 32GB desktop. Although statm gives totals in 7 classes, two are not used and others, particularly the number of data pages and pages occupied by the stack, are "broken" [77, Table 1-3] and difficult to interpret.

2) Massif high overhead but gives peak heap use in bytes: Valgrind's performance tuning tool massif, can report both dynamic and peak heap usage with high precision. That is, it reports the actual number of bytes, rather than the number of 4KB pages used. However massif imposes, particularly if very accurate results are wanted, a considerable overhead, slowing down gem5 by about 15 fold.

3) Google's TCmalloc heap manager: It is also possible to build gem5 with Google's TCmalloc heap manager[3]. However TCmalloc is not compatible with glibc's measurement tools and statm (see Section II-C1) results are dominated by the increased executable file size so confusing comparisons with glibc. Nonetheless TCmalloc may be an interesting approach.

4) gem5 malloc_info used with Magpie and CMA-ES: Both statm (Section II-C1) and massif (Section II-C2) have the advantage of being general and can be applied without code changes whereas we had to modify gem5 to insert a call to malloc_info. We decided to use malloc_info as it is accurate to the individual byte level, it has only a little noise, it imposes almost no additional runtime overhead, and it relates directly to the parameters we are tuning.

---

[2]We use the release of gem5 produced by Bobby R. Bruce for the SSBSE 2023 challenge track [74].

[3]https://google.github.io/tcmalloc/

TABLE I
GNU GLIBC 2.17 MALLOC TUNABLE PARAMETERS

name	default	notes
M_MMAP_MAX	65536	Limit on mmap use
M_MMAP_THRESHOLD	131072	When to use mmap
M_TRIM_THRESHOLD	131072	When to return memory to the operating system
M_PERTURB	0	debug aid
M_TOP_PAD	0	Only multi-threaded
M_ARENA_TEST	0	Only multi-threaded
M_ARENA_MAX	0	Only multi-threaded

TABLE II
MAGPIE PARAMETER FILE MALLOPT30.TXT (TABLE I)

```
TIMING="run"
M_MMAP_MAX_tune        g[0,33554432][65536]
M_TRIM_THRESHOLD_tune  g[0,33554432][131072]
M_MMAP_THRESHOLD_tune  g[0,33554432][131072]
CLI_PREFIX = ""
CLI_GLUE = "="
```

D. gem5 glibc Parameter Search Space

There are 7 GNU malloc tuning parameters (see Table I). These each take non-negative integer values. Giving a search space of $(2^{31})^7 = 2.1\ 10^{65}$, clearly beyond enumeration. However a quick inspection of the documentation[4] reveals three parameters only apply to multi-threaded programs, whereas gem5 is single threaded and another is provided as a debug aid. This leaves three parameters, giving a search space of $(2^{31})^3 = 9.9\ 10^{27}$, which is still not enumerable. Although the documentation says the very act of changing one of the 3 remaining parameters, M_MMAP_THRESHOLD, from its default value, causes malloc etc. to change their behaviour, Figure 1 shows the impact of explicitly setting the two most important tunable parameters.

III. MAGPIE

Magpie (Machine Automated General Performance Improvement via Evolution of software) [52] is an open source language independent genetic improvement tool written by Aymeric Blot at UCL. It was first released in 2022[5]. It is often used to improve C and Java source code but also has the ability to tune parameters associated with programs. The Magpie user specifies the parameters to be optimised in a parameter file (see example in Table II). The file specifies with each parameter: its name, type (categorical or continuous) range and associated probability distribution (uniform, exponential, uniform discrete or geometric, but see also Section III-C) and its default value. It is also possible to provide information on legal and illegal combinations of parameters [78].

A. Magpie Parameter Search for gem5's heap

To tune malloc glibc parameters to minimise gem5's total heap, Magpie was run in local search mode (`python3 -m bin.local_search --scenario examples/scenario/mallopt30.txt`) using g++ version 10.2.1 and Python 3.7.3 for 1000 steps using Magpie's defaults. Notice (Table II) the search space has been further reduced (from $(2^{31})^7$ Section II-D) to $(1 + 2^{25})^3 = 33\,554\,433^3 = 37\,778\,935\,240\,656\,982\,900\,736 = 3.78\ 10^{22}$, by restricting the range of values Magpie can explore for all three parameters to zero plus the positive integers up to 2^{25}.

The value 2^{25} was chosen as being 256 times bigger than the largest default value (column 2 in Table I) and so probably much bigger than needed. In Table II the suffix `_tune` is appended to the glibc parameter names only to avoid any confusion between the mechanism being used to tune glibc and the actual parameter itself. (The data in Figure 1 was obtained by interpolating between 2D grid points with a typical ratio of $\sqrt{2}$ spanning 0 to 33 554 432.)

B. glibc malloc_info Fitness Function

The only change needed to gem5 was to insert a call to gather heap usage statistic by calling malloc_info just before gem5 finishes. (For convenience and debugging the code also reports glibc heap parameters, statm (Section II-C1) and uses Linux perf [79], [80], [70] to gather statistics on run time.) After gem5 finishes an external script extracts the size of the conventional heap and mmap memory from malloc_info's output and adds them to give the actual number of bytes of memory used by gem5's heap (including overheads, Section II-C). This sum becomes the fitness which Magpie and CMA-ES try to minimise.

Notice we are only calling malloc_info at the end of the gem5 run. This works well with gem5 but may be less helpful with other programs. Valgrind's massif (Section II-C2) confirms this gives peak heap use for gem5.

Genetic improvement is often used to speed up software, when it is critical to measure how fast each mutant is compared to the original code. However measuring run time is notoriously noisy and typically multiple measurements are used and an average taken. This is what we do here, even though malloc_info is typically more consistent than run time (malloc_info noise $< 0.05\%$).

gem5 is first run without changing any of the glibc malloc parameters (Table I). All its outputs are saved and the size of its heap is recorded. Then gem5 is run again with the mutated values of M_MMAP_MAX, M_MMAP_THRESHOLD and M_TRIM_THRESHOLD. Again its outputs and heap size are recorded.

Both gem5 runs have a timeout of 15 seconds and identical command lines (`gem5 $script --isa X86 --binary $binary`). Where `$script` is a Python script used to control the gem5 run, e.g. the type of CPU to simulate, the type and size of the memory, characteristics of the link between the two, etc. Like [81], we use `hello-custom-binary.py` which was supplied with the SSBSE 2023 challenge track [74].

---

[4]https://www.gnu.org/software/libc/manual/html_node/Malloc-Tunable-Parameters.html

[5]We use Magpie downloaded on 2 October 2023 https://github.com/bloa/magpie.

`$binary` is a file containing the machine code whose execution gem5 is to simulate. For example, it might be a performance critical code segment which a gem5 user wishes to simulate in order to tune hardware before manufacture or to diagnose performance issues in existing designs. We use a self contained compiled C fragment taken from the performance critical loop in RNAfold [82]–[84][6]. Our binary is set up to execute exactly, including the actual RNA data values, 209 times the innermost actions of RNAfold folding an RNA molecule composed of twenty bases ($209 = \frac{1}{2}(20+1)20-1$). (After Magpie had been run, its output was verified with gem5 simulating a verity of binaries created by compiling different C programs.)

Each output from the two gem5 runs is compared. If any output were to be different the whole fitness evaluation would be aborted. This did not happen in any production run. Even though there is little noise, for convenience, we follow recent practise [54] and repeat this eleven times for pairs of gem5 runs. If all 22 gem5 runs are successful, the fitness of the mutant returned to Magpie (or CMA-ES) is the median of the 11 pairs of runs of the ratio of the heap size with the mutated parameters and without. If the parameters cause gem5 to timeout a very poor fitness value ($100\times$ the default) is returned. Other errors are fatal, and after debugging did not occur. I.e., all gem5 runs gave the right answers, except when badly mistuned malloc parameters cause it to slowed down so much that it was timed out.

C. Magpie's Exploration of the Search Space

Figure 2 shows the progress of ten independent Magpie runs. All the Magpie runs made some progress (orange dots), but one run (run 0 +) succeeded in reducing M_MMAP_THRESHOLD from its default value 131 072 to 268 in fitness evaluation 388, which lies comfortably in the high fitness yellow valley of Figure 1 (fitness 11.1%). The best fitness in Magpie run 0 was found in fitness evaluation 416 (also M_MMAP_THRESHOLD = 268) and gives a reduction in gem5's heap of 11.5% (The best value found by our grid Figure 1 occurs at $2^{8.0} = 256$ and is 11.9%. Earlier, ad hoc Magpie runs, found savings of up to 12.5%.)

Figure 3 concentrates upon another Magpie run (run 1) and includes all three malloc parameters. Figure 3 highlights the way Magpie searches parallel to each of the three axis. That is, as Magpie finds new preferred settings it randomly changes each of the three parameters one at a time leading to lines of sample points (blue dots in Figure 3). E.g., notice the vertical lines of blue dots due to sampling with different values of M_MMAP_MAX but leaving the other parameters unchanged. Similarly mutations to M_TRIM_THRESHOLD give lines of blue dots parallel to the y-axis. The black arrows in Figure 3 highlight each new best fitness so far. The arrows show that often Magpie makes progress by changing one parameter at a time but multiple changes are also used.

---

[6]RNAfold is the state of the art bioinformatics tool for minimising free energy in RNA molecules. It is a large open source project written in C https://www.tbi.univie.ac.at/RNA/RNAfold.1.html.
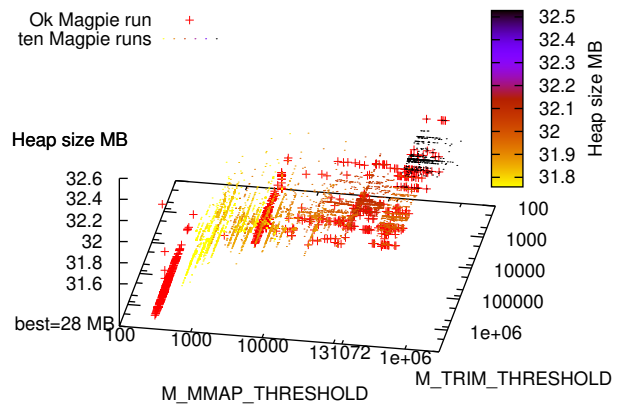

Fig. 2. Ten Magpie runs. Successful run plotted with +, with 9 runs making little progress (orange $\approx$ 1% better). To avoid compressing vertical axis, "best" values are not to scale and timeouts are not plotted. (Runs 3 and 6 each had one sample which drastically slowed gem5, causing Magpie to time it out. The values were M_MMAP_THRESHOLD=25 and =30.) Non-linear scales.
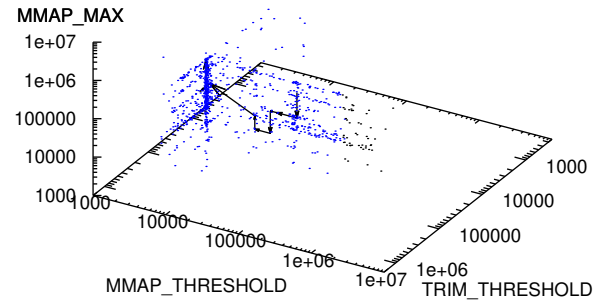

Fig. 3. Typical Magpie run. Arrows highlight best point found so far. (All runs start at default 128K,128K,65336.) This run reduces mmap but gets stuck near mmap=8356 and so improves by only 1%. Noise added to spread data. Note log scales.

Figure 4 shows the distribution of M_MMAP_THRESHOLD values sampled in ten Magpie runs (note log scale). On average, the 268 *new* values sampled are exponentially distributed with a mean of 340 000 (as expected this is 1% of the range of M_MMAP_THRESHOLD values, see Table II). However where Magpie found parameter settings which reduce gem5's heap, there are appreciable peaks due to M_MMAP_THRESHOLD values being reused. These are specific to each run. Notice despite finding lower better values, due to the large mean, Magpie continues to preferentially sample large values. Crucially only Magpie run 0 samples values of M_MMAP_THRESHOLD from the 255–511 bin, corresponding roughly to the region giving heap saving of more than 10% shown as the yellow valley in Figure 1.

IV. CMA-ES

We also tried Hansen's Covariance Matrix Adaptation Evolution Strategy (CMA-ES)[7]algorithm [55]. We used the C version of CMA-ES from Krauss' [85]–[89] replication package [90] compiled with GCC version 7.3.1. Although
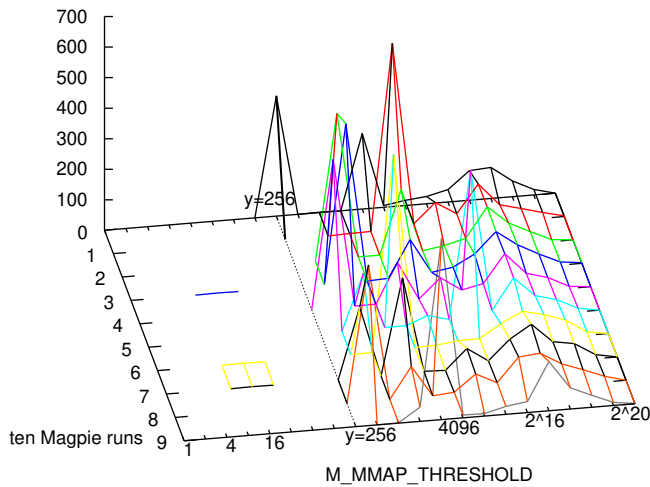
---

[7] https://cma-es.github.io/

Fig. 4. Distribution of M_MMAP_THRESHOLD values in ten Magpie runs. Bins cover factor of 2.0. Dotted line at y=256 highlights only run 0 sampled M_MMAP_THRESHOLD in the range 256–511. Note log scale.
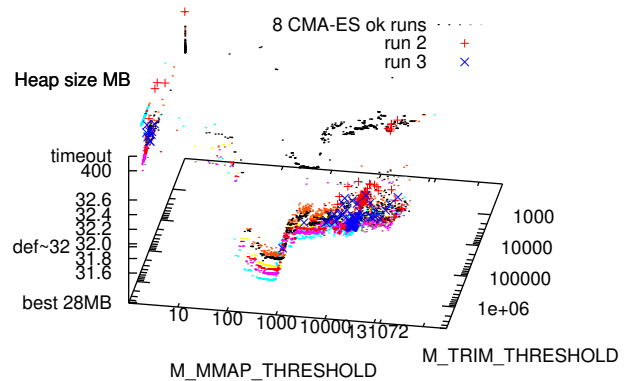

Fig. 5. Ten CMA-ES runs. Eight successful runs shown with coloured dots. Two runs which found only 0.5% improvement plotted with + and ×. To avoid compressing vertical axis, small values and values 32.6–400 megabytes plotted at reduced scales. For log scales, non-positive values plotted at edges. Timeouts plotted above all legal heap values.

Hansen [91] recommends preconditioning, the data did not need careful preparation and we simply re-used the Magpie fitness function. That is, we used the natural data ranges for all three parameters. However we did round the `double` values in the CMA-ES population to integers in the same range as Magpie (i.e. $0 \cdots 2^{25}$). Similarly CMA-ES was initialised with the glibc default values and its initial standard deviations were set to 25% of these start values [91]. Otherwise CMA-ES's defaults were used and like Magpie it was run for 1 000 fitness evaluations. (Note although CMA-ES uses a 3×64 bit double precision representation, by rounding to the same integer ranges as we used with Magpie, the CMA-ES search space is the same as the Magpie one, i.e. $3.78\ 10^{22}$.) Since they use the same fitness function and number of evaluations, CMA-ES and Magpie runtimes are similar. (Magpie is about 10% faster because where it resamples a point in the search space, it does not re-evaluate it but instead uses its cache of previously measured fitness values.)

Of ten independent CMA-ES runs (see Figure 5), two runs find improvements of only 0.5% (+ and × in Figure 5). For example, early in run 2, CMA-ES overshoots the sweet spot and tests negative parameter settings which sometimes cause gem5 to be timed out (vertical values > 400 plotted with + at left edge in Figure 5). Thus in run 2 CMA-ES learns to avoid small values and in the last 80% of run 2 CMA_ES almost never samples M_MMAP_THRESHOLD below 10 000. Similarly, although CMA-ES run 3 (× in Figure 5) does not get as many timeouts, early in the run it also repeatedly samples negative values (which are mapped to M_MMAP_THRESHOLD = 0) which have poor fitness (black region in Figure 1), and again in run 3 CMA-ES learns to avoid small values and so in the last 60% of run 3 CMA-ES almost never samples M_MMAP_THRESHOLD below 10 000. Note the similarity between Figure 1 and Langdon and Poli's synthetic benchmark [92, Fig. 23], which was deliberately evolved to be difficult for CMA-ES. However the glibc sweet spot is large enough that,

although in the other eight runs CMA-ES also overshoots and also occasionally causes gem5 to timeout, it does not shun small values (dots in Figure 5) and later in these runs it does home in on the sweet spot. Therefore CMA-ES is able to find glibc parameter settings which reduced gem5's heap by on average 11.6% (best 11.7%), essentially by setting M_MMAP_THRESHOLD to a value between 338 and 520 (best 430).

V. Conclusions

We have shown that both Magpie and CMA-ES can be applied out of the box to a non-trivial software engineering problem (reducing by at least 11% the heap of a program composed of more than a million lines of C++ without changing its code or slowing it down) in less than 14 hours (total 10×1000 fitness evaluations on an 8 core 3.6 GHz desktop) despite the fact that the search space is 37 778 935 240 656 982 900 736 $(3.78\ 10^{22})$.

We conclude, in software engineering, size is not necessarily a guide to problem difficulty and sometimes SE problems are easier than they might appear.

References

[1] S. Wright, "The roles of mutation, inbreeding, crossbreeding and selection in evolution," in *Proceedings of the Sixth Annual Congress of Genetics*. Ithaca, NY, USA: The Genetics Society of America, 1932, pp. 356–366. [Online]. Available: http://www.blackwellpublishing.com/ridley/classictexts/wright.pdf

[2] T. Jones, "One operator, one landscape," Santa Fe Institute, Tech. Rep. SFI TR 95-02-025, January 1995. [Online]. Available: https://www.santafe.edu/research/results/working-papers/one-operator-one-landscape

[3] C. R. Reeves and J. E. Rowe, *Genetic Algorithms–Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Publishers, 2003. [Online]. Available: http://dx.doi.org/10.1007/b101880

[4] P. F. Stadler, "Fitness landscapes," in *Biological Evolution and Statistical Physics*, M. Laessig and A. Valleriani, Eds. Springer, 2002, pp. 183–204. [Online]. Available: http://dx.doi.org/10.1007/3-540-45692-9_10

[5] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. Springer-Verlag, 2002. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-04726-2

[6] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, Dec. 2001. [Online]. Available: http://dx.doi.org/10.1016/S0950-5849(01)00189-6

[7] W. B. Langdon, "Genetic improvement of programs," in *18th International Conference on Soft Computing, MENDEL 2012*, R. Matousek, Ed. Brno, Czech Republic: Brno University of Technology, 27-29 Jun. 2012, invited keynote. [Online]. Available: http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Langdon_2012_mendel.pdf

[8] W. B. Langdon and M. Harman, "Optimising existing software with genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2013.2281544

[9] J. Petke, W. B. Langdon, and M. Harman, "Applying genetic improvement to MiniSAT," in *Symposium on Search-Based Software Engineering*, ser. Lecture Notes in Computer Science, G. Ruhe and Yuanyuan Zhang, Eds., vol. 8084. Leningrad: Springer, Aug. 24-26 2013, pp. 257–262, short Papers. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39742-4_21

[10] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic improvement of software: a comprehensive survey," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 415–432, Jun. 2018. [Online]. Available: http://dx.doi.org/doi:10.1109/TEVC.2017.2693219

[11] A. Blot and J. Petke, "Empirical comparison of search heuristics for genetic improvement of software," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 5, pp. 1001–1011, Oct. 2021. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2021.3070271

[12] E. Schulte, J. DiLorenzo, W. Weimer, and S. Forrest, "Automated repair of binary and assembly programs for cooperating embedded devices," in *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS 2013. Houston, Texas, USA: ACM, Mar. 16-20 2013, pp. 317–328. [Online]. Available: http://dx.doi.org/10.1145/2451116.2451151

[13] E. Schulte, Zachary P. Fry, E. Fast, W. Weimer, and S. Forrest, "Software mutational robustness," *Genetic Programming and Evolvable Machines*, vol. 15, no. 3, pp. 281–312, Sep. 2014. [Online]. Available: http://dx.doi.org/10.1007/s10710-013-9195-8

[14] E. Schulte, "Neutral networks of real-world programs and their application to automated software evolution," Ph.D. dissertation, University of New Mexico, Albuquerque, USA, Jul. 2014. [Online]. Available: https://digitalrepository.unm.edu/cs_etds/49/

[15] E. Schulte, W. Weimer, and S. Forrest, "Repairing COTS router firmware without access to source code or test suites: A case study in evolutionary software repair," in *Genetic Improvement 2015 Workshop*, W. B. Langdon, J. Petke, and D. R. White, Eds. Madrid: ACM, 11-15 Jul. 2015, pp. 847–854, best Paper. [Online]. Available: http://dx.doi.org/10.1145/2739482.2768427

[16] M. Orlov and M. Sipper, "Flight of the FINCH through the Java wilderness," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 166–182, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2010.2052622

[17] K. Yeboah-Antwi and B. Baudry, "Embedding adaptivity in software systems using the ECSELR framework," in *Genetic Improvement 2015 Workshop*, W. B. Langdon, J. Petke, and D. R. White, Eds. Madrid: ACM, 11-15 Jul. 2015, pp. 839–844. [Online]. Available: http://dx.doi.org/10.1145/2739482.2768425

[18] E. Schulte, S. Forrest, and W. Weimer, "Automated program repair through the evolution of assembly code," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. Antwerp: ACM, 20-24 Sep. 2010, pp. 313–316. [Online]. Available: http://dx.doi.org/10.1145/1858996.1859059

[19] J.-Y. Liou, S. Forrest, and Carole-Jean Wu, "Genetic improvement of GPU code," in *GI-2019, ICSE workshops proceedings*, J. Petke, Shin Hwei Tan, W. B. Langdon, and W. Weimer, Eds. Montreal: IEEE, 28 May 2019, pp. 20–27, Best Paper. [Online]. Available: http://dx.doi.org/10.1109/GI.2019.00014

[20] W. B. Langdon, A. Al-Subaihin, A. Blot, and D. Clark, "Genetic improvement of LLVM intermediate representation," in *EuroGP 2023: Proceedings of the 26th European Conference on Genetic Programming*, ser. LNCS, G. Pappa, M. Giacobini, and Z. Vasicek, Eds., vol. 13986. Brno, Czech Republic: Springer Verlag, 12-14 Apr. 2023, pp. 244–259. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-29573-7_16

[21] Z. Nemeth, P. Faulkner Rainford, and B. Porter, "Phenotypic species definitions for genetic improvement of source code," in *ALIFE 2024: Proceedings of the 2024 Artificial Life Conference*, A. Faina, S. Risi, E. Medvet, K. Stoy, B. Chan, K. Miras, P. Zahadat, D. Grbic, and G. Nadizar, Eds., The International Society for Artificial Life. Copenhagen: MIT Press, Jul. 22-26 2024, pp. 530–539. [Online]. Available: http://dx.doi.org/10.1162/isal_a_00795

[22] G. O'Brien and J. Clark, "Using genetic improvement to retarget quantum software on differing hardware," in *GI @ ICSE 2021*, J. Petke, B. R. Bruce, Y. Huang, A. Blot, W. Weimer, and W. B. Langdon, Eds. internet: IEEE, 30 May 2021, pp. 31–38, winner Best Presentation. [Online]. Available: http://dx.doi.org/10.1109/GI52543.2021.00015

[23] A. Murphy, T. Laurent, and A. Ventresque, "The case for grammatical evolution in test generation," in *GI @ GECCO 2022*, B. R. Bruce, V. Nowack, A. Blot, E. Winter, W. B. Langdon, and J. Petke, Eds. Boston, USA: Association for Computing Machinery, 9 Jul. 2022, pp. 1946–1947. [Online]. Available: http://dx.doi.org/10.1145/3520304.3534042

[24] M. Lorandi, L. L. Custode, and G. Iacca, "Genetic improvement of routing protocols for delay tolerant networks," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 1, May 2021. [Online]. Available: http://dx.doi.org/10.1145/3453683

[25] E. Fredericks and B. DeVries, "(genetically) improving novelty in procedural story generation," in *GI @ ICSE 2021*, J. Petke, B. R. Bruce, Y. Huang, A. Blot, W. Weimer, and W. B. Langdon, Eds. internet: IEEE, 30 May 2021, pp. 39–40. [Online]. Available: http://dx.doi.org/10.1109/GI52543.2021.00016

[26] Y. Huang, H. Ahmad, S. Forrest, and W. Weimer, "Applying automated program repair to dataflow programming languages," in *GI @ ICSE 2021*, J. Petke, B. R. Bruce, Y. Huang, A. Blot, W. Weimer, and W. B. Langdon, Eds. internet: IEEE, 30 May 2021, pp. 21–22. [Online]. Available: http://dx.doi.org/10.1109/GI52543.2021.00013

[27] I. Mesecan, M. C. Gerten, J. I. Lathrop, M. B. Cohen, and T. Haddad Caldas, "CRNRepair: Automated program repair of chemical reaction networks," in *GI @ ICSE 2021*, J. Petke, B. R. Bruce, Y. Huang, A. Blot, W. Weimer, and W. B. Langdon, Eds. internet: IEEE, 30 May 2021, pp. 23–30, best paper. [Online]. Available: http://dx.doi.org/10.1109/GI52543.2021.00014

[28] J. Callan and J. Petke, "Optimising SQL queries using genetic improvement," in *GI @ ICSE 2021*, J. Petke, B. R. Bruce, Y. Huang, A. Blot, W. Weimer, and W. B. Langdon, Eds. internet: IEEE, 30 May 2021, pp. 9–10. [Online]. Available: http://dx.doi.org/10.1109/GI52543.2021.00010

[29] F. Molina, P. Ponzio, N. Aguirre, and M. Frias, "EvoSpex: An evolutionary algorithm for learning postconditions (artifact)," in *IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, S. Abrahao and D. Mendez, Eds., 25-28 May 2021, pp. 186–186. [Online]. Available: http://dx.doi.org/10.1109/ICSE-Companion52605.2021.00080

[30] F. de A. Farzat, M. de O. Barros, and G. H. Travassos, "Evolving JavaScript code to reduce load time," *IEEE Transactions on Software Engineering*, vol. 47, no. 8, pp. 1544–1558, Aug. 2021. [Online]. Available: http://dx.doi.org/10.1109/TSE.2019.2928293

[31] I. Mesecan, D. Blackwell, D. Clark, M. B. Cohen, and J. Petke, "HyperGI: Automated detection and repair of information flow leakage," in *The 36th IEEE/ACM International Conference on Automated Software Engineering, New Ideas and Emerging Results track, ASE NIER 2021*, H. Khalajzadeh and J.-G. Schneider, Eds., Melbourne, 15-19 Nov. 2021, pp. 1358–1362. [Online]. Available: http://dx.doi.org/10.1109/ASE51524.2021.9678758

[32] I. Kosorukov, D. Blackwell, D. Clark, M. Cohen, and J. Petke, "Mining for mutation operators for reduction of information flow control violations," in *IEEE/ACM International Conference on Automated Software Engineering, The New Ideas and Emerging Results (ASE-NIER 2024)*, Sacramento, 24 Oct. 27-Nov. 1 2024. [Online]. Available: http://dx.doi.org/10.1145/3691620.3695308

[33] B. R. Bruce, "The blind software engineer: Improving the non-functional properties of software by means of genetic improvement," Ph.D. dissertation, Computer Science, University College, London,

UK, 12 Jul. 2018. [Online]. Available: http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/bruce_bobby_r_thesis.pdf

[34] N. Burles, E. Bowles, B. R. Bruce, and K. Srivisut, "Specialising Guava's cache to reduce energy consumption," in *SSBSE*, ser. LNCS, Y. Labiche and M. Barros, Eds., vol. 9275. Bergamo, Italy: Springer, Sep. 5-7 2015, pp. 276–281. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-22183-0_23

[35] B. R. Bruce, J. Petke, M. Harman, and E. T. Barr, "Approximate oracles and synergy in software energy search spaces," *IEEE Transactions on Software Engineering*, vol. 45, no. 11, pp. 1150–1169, Nov. 2019. [Online]. Available: http://dx.doi.org/10.1109/TSE.2018.2827066

[36] J. Callan, W. B. Langdon, and J. Petke, "On reducing network usage with genetic improvement," in *13th International Workshop on Genetic Improvement @ICSE 2024*, Gabin An, A. Blot, V. Nowack, O. Krauss, and J. Petke, Eds. Lisbon: ACM, 16 Apr. 2024, pp. 23–30. [Online]. Available: http://dx.doi.org/10.1145/3643692.3648262

[37] Fan Wu, W. Weimer, M. Harman, Yue Jia, and J. Krinke, "Deep parameter optimisation," in *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, S. Silva *et al.*, Eds. Madrid: ACM, 11-15 Jul. 2015, pp. 1375–1382. [Online]. Available: http://dx.doi.org/10.1145/2739480.2754648

[38] Fan Wu, "Mutation-based genetic improvement of software," Ph.D. dissertation, Department of Computer Science, University College, London, UK, Jul. 2 2017. [Online]. Available: http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Thesis_Fan_v2.1.pdf

[39] W. B. Langdon and M. Harman, "Evolving a CUDA kernel from an nVidia template," in *2010 IEEE World Congress on Computational Intelligence*, P. Sobrevilla, Ed. Barcelona: IEEE, 18-23 Jul. 2010, pp. 2376–2383. [Online]. Available: http://dx.doi.org/10.1109/CEC.2010.5585922

[40] ——, "Genetically improved CUDA C++ software," in *17th European Conference on Genetic Programming*, ser. LNCS, M. Nicolau, K. Krawiec, M. I. Heywood, M. Castelli, P. Garcia-Sanchez, J. J. Merelo, V. M. Rivas Santos, and K. Sim, Eds., vol. 8599. Granada, Spain: Springer, 23-25 Apr. 2014, pp. 87–99. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44303-3_8

[41] W. B. Langdon, Brian Yee Hong Lam, M. Modat, J. Petke, and M. Harman, "Genetic improvement of GPU software," *Genetic Programming and Evolvable Machines*, vol. 18, no. 1, pp. 5–44, Mar. 2017. [Online]. Available: http://dx.doi.org/10.1007/s10710-016-9273-9

[42] B. R. Bruce and J. Petke, "Towards automatic generation and insertion of OpenACC directives," University College, London, London, UK, Tech. Rep. RN/18/04, 12 Apr. 2018. [Online]. Available: http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research_Notes/RN_18_04.pdf

[43] A. Marginean, E. T. Barr, M. Harman, and Y. Jia, "Automated transplantation of call graph and layout features into Kate," in *SSBSE*, ser. LNCS, Y. Labiche and M. Barros, Eds., vol. 9275. Bergamo, Italy: Springer, Sep. 5-7 2015, pp. 262–268, winner Gold HUMIE. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-22183-0_21

[44] A. Marginean, "Automated software transplantation," Ph.D. dissertation, University College London, UK, 8 Nov. 2021, ACM SIGEVO Award for the best dissertation of the year. [Online]. Available: https://discovery.ucl.ac.uk/id/eprint/10137954/

[45] M. Zamorano, A. Domingo, C. Cetina, and F. Sarro, "Game software engineering: A controlled experiment comparing automated content generation techniques," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Barcelona, Spain, October 24-25 2024, best paper. [Online]. Available: http://dx.doi.org/10.1145/3674805.3686690

[46] B. R. Bruce, "Automatically exploring computer system design spaces," in *GI @ GECCO 2022*, B. R. Bruce, V. Nowack, A. Blot, E. Winter, W. B. Langdon, and J. Petke, Eds. Boston, USA: Association for Computing Machinery, 9 Jul. 2022, pp. 1926–1927. [Online]. Available: http://dx.doi.org/10.1145/3520304.3534021

[47] G. Jahangirova, D. Clark, M. Harman, and P. Tonella, "An empirical validation of oracle improvement," *IEEE Trans. Software Eng.*, vol. 47, no. 8, pp. 1708–1728, 2021. [Online]. Available: http://dx.doi.org/10.1109/TSE.2019.2934409

[48] A. E. I. Brownlee, J. Callan, K. Even-Mendoza, A. Geiger, C. Hanna, J. Petke, F. Sarro, and D. Sobania, "Enhancing genetic improvement mutations using large language models," in *SSBSE 2023: Challenge Track*, ser. LNCS, P. Arcaini, Tao Yue, and E. Fredericks, Eds., vol. 14415. San Francisco, USA: Springer, 8 Dec. 2023, pp. 153–159. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-48796-5_13

[49] G. Pinna, D. Ravalico, L. Rovito, L. Manzoni, and A. De Lorenzo, "Enhancing large language models-based code generation by leveraging genetic improvement," in *EuroGP 2024: Proceedings of the 27th European Conference on Genetic Programming*, ser. LNCS, M. Giacobini, Bing Xue, and L. Manzoni, Eds., vol. 14631. Aberystwyth: Springer, 3-5 Apr. 2024, pp. 108–124. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-56957-9_7

[50] D. R. White, "GI in no time," in *GI-2017*, J. Petke, D. R. White, W. B. Langdon, and W. Weimer, Eds. Berlin: ACM, 15-19 Jul. 2017, pp. 1549–1550. [Online]. Available: http://dx.doi.org/doi:10.1145/3067695.3082515

[51] A. E. I. Brownlee, J. Petke, B. Alexander, E. T. Barr, M. Wagner, and D. R. White, "Gin: genetic improvement research made easy," in *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference*, M. Lopez-Ibanez *et al.*, Eds. Prague, Czech Republic: ACM, 13-17 Jul. 2019, pp. 985–993. [Online]. Available: http://dx.doi.org/10.1145/3321707.3321841

[52] A. Blot and J. Petke, "MAGPIE: Machine automated general performance improvement via evolution of software," arXiv, 4 Aug. 2022. [Online]. Available: http://dx.doi.org/10.48550/arxiv.2208.02811

[53] W. B. Langdon, Brian Yee Hong Lam, J. Petke, and M. Harman, "Improving CUDA DNA analysis software with genetic programming," in *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, S. Silva *et al.*, Eds. Madrid: ACM, 11-15 Jul. 2015, pp. 1063–1070. [Online]. Available: http://dx.doi.org/10.1145/2739480.2754652

[54] W. B. Langdon and D. Clark, "Genetic improvement of last level cache," in *EuroGP 2024: Proceedings of the 27th European Conference on Genetic Programming*, ser. LNCS, M. Giacobini, Bing Xue, and L. Manzoni, Eds., vol. 14631. Aberystwyth: Springer Verlag, 3-5 Apr. 2024, pp. 209–226. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-56957-9_13

[55] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, Summer 2001. [Online]. Available: http://dx.doi.org/10.1162/106365601750190398

[56] J. Petke, B. Alexander, E. T. Barr, A. E. I. Brownlee, M. Wagner, and D. R. White, "Program transformation landscapes for automated program modification using Gin," *Empirical Software Engineering*, vol. 28, p. article no: 104, 2023, accepted to the journal-first track at ICSE 2024. [Online]. Available: http://dx.doi.org/10.1007/s10664-023-10344-5

[57] W. B. Langdon, "The genetic improvement fitness landscape," University College, London, London, UK, Tech. Rep. RN/16/04, 10 Jun. 2016. [Online]. Available: http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research_Notes/rn1604.pdf

[58] W. B. Langdon and M. Harman, "Fitness landscape of the Triangle program," in *PPSN-2016 Workshop on Landscape-Aware Heuristic Search*, N. Veerapen and G. Ochoa, Eds., Edinburgh, 17 Sep. 2016, also available as UCL RN/16/05. [Online]. Available: http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research_Notes/rn1605.pdf

[59] W. B. Langdon, N. Veerapen, and G. Ochoa, "Visualising the search landscape of the Triangle program," in *EuroGP 2017*, ser. LNCS, M. Castelli, J. McDermott, and L. Sekanina, Eds., vol. 10196. Amsterdam: Springer, 19-21 Apr. 2017, pp. 96–113. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-55696-3_7

[60] N. Veerapen, F. Daolio, and G. Ochoa, "Modelling genetic improvement landscapes with local optima networks," in *GI-2017*, J. Petke, D. R. White, W. B. Langdon, and W. Weimer, Eds. Berlin: ACM, 15-19 Jul. 2017, pp. 1543–1548, best presentation prize. [Online]. Available: http://dx.doi.org/10.1145/3067695.3082518

[61] N. Veerapen and G. Ochoa, "Visualising the global structure of search landscapes: genetic improvement as a case study," *Genetic Programming and Evolvable Machines*, vol. 19, no. 3, pp. 317–349, Sep. 2018, special issue on genetic programming, evolutionary computation and visualization. [Online]. Available: http://dx.doi.org/10.1007/s10710-018-9328-1

[62] J. Renzullo, W. Weimer, M. Moses, and S. Forrest, "Neutrality and epistasis in program space," in *GI-2018, ICSE workshops proceedings*, J. Petke, K. Stolee, W. B. Langdon, and W. Weimer, Eds. Gothenburg, Sweden: ACM, 2 Jun. 2018, pp. 1–8, best presentation award. [Online]. Available: http://dx.doi.org/10.1145/3194810.3194812

[63] J. Renzullo, W. Weimer, and S. Forrest, "Evolving software: Combining online learning with mutation-based stochastic search,"

ACM Transactions on Evolutionary Learning and Optimization, vol. 3, no. 4, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1145/3597617

[64] Chunyan Mu and D. Clark, "An interval-based abstraction for quantifying information flow," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 3, pp. 119–141, 2009, proceedings of Seventh Workshop on Quantitative Aspects of Programming Languages (QAPL 2009). [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2009.10.009

[65] K. Androutsopoulos, D. Clark, Haitao Dan, R. M. Hierons, and M. Harman, "An analysis of the relationship between conditional entropy and failed error propagation in software testing," in 36th *International Conference on Software Engineering (ICSE 2014)*, L. Briand and A. van der Hoek, Eds. Hyderabad, India: ACM, 31 May- 7 June 2014, pp. 573–583. [Online]. Available: http://dx.doi.org/10.1145/2568225.2568314

[66] J. Petke, D. Clark, and W. B. Langdon, "Software robustness: A survey, a theory, and some prospects," in *ESEC/FSE 2021, Ideas, Visions and Reflections*, P. Avgeriou and Dongmei Zhang, Eds. Athens, Greece: ACM, 23-28 Aug. 2021, pp. 1475–1478. [Online]. Available: http://dx.doi.org/10.1145/3468264.3473133

[67] W. B. Langdon and J. Petke, "Software is not fragile," in *Complex Systems Digital Campus E-conference, CS-DC'15*, ser. Proceedings in Complexity, P. Parrend, P. Bourgine, and P. Collet, Eds. Springer, Sep. 30-Oct. 1 2015, pp. 203–211, invited talk. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-45901-1_24

[68] J. M. Voas, "PIE: a dynamic failure-based technique," *IEEE Transactions on Software Engineering*, vol. 18, no. 8, pp. 717–727, Aug 1992. [Online]. Available: http://dx.doi.org/10.1109/32.153381

[69] W. B. Langdon and D. Clark, "Deep mutations have little impact," in *13th International Workshop on Genetic Improvement @ICSE 2024*, Gabin An, A. Blot, V. Nowack, O. Krauss, and J. Petke, Eds. Lisbon: ACM, 16 Apr. 2024, pp. 1–8, best paper. [Online]. Available: http://dx.doi.org/10.1145/3643692.3648259

[70] ——, "Deep imperative mutations have less impact," *Automated Software Engineering*, vol. 32, p. article number 6, 2025. [Online]. Available: http://dx.doi.org/10.1007/s10515-024-00475-4

[71] W. B. Langdon, J. Petke, and R. Lorenz, "Evolving better RNAfold structure prediction," in *EuroGP 2018: Proceedings of the 21st European Conference on Genetic Programming*, ser. LNCS, M. Castelli, L. Sekanina, and Mengjie Zhang, Eds., vol. 10781. Parma, Italy: Springer Verlag, 4-6 Apr. 2018, pp. 220–236. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-77553-1_14

[72] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. S. B. Altaf, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, May 2011. [Online]. Available: http://dx.doi.org/10.1145/2024716.2024718

[73] B. R. Bruce, A. Akram, H. Nguyen, K. Roarty, M. Samani, M. Fariborz, T. Reddy, M. D. Sinclair, and J. Lowe-Power, "Enabling reproducible and agile full-system simulation," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS*. Stony Brook, NY, USA: , March 28-30 2021, pp. 183–193. [Online]. Available: http://dx.doi.org/10.1109/ISPASS51385.2021.00035

[74] P. Arcaini, T. Yue, and E. M. Fredericks, Eds., *Search-Based Software Engineering - 15th International Symposium, SSBSE 2023, Proceedings*, ser. Lecture Notes in Computer Science, vol. 14415. San Francisco, USA: Springer, December 8 2023. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-48796-5

[75] J. Umeike, N. Patel, A. Manley, A. Mamandipoor, Heechul Yun, and M. Alian, "Profiling gem5 simulator," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Raleigh, NC, USA, 23-25 April 2023, pp. 103–113. [Online]. Available: http://dx.doi.org/10.1109/ISPASS57527.2023.00019

[76] A. Dakhama, K. Even-Mendoza, W. B. Langdon, H. Menendez Benito, and J. Petke, "SearchGEM5: Towards reliable gem5 with search based software testing and large language models," in *SSBSE 2023: Challenge Track*, ser. LNCS, P. Arcaini, Tao Yue, and E. Fredericks, Eds., vol. 14415. San Francisco, USA: Springer, 8 Dec. 2023, pp. 60–166, winner best challenge track paper. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-48796-5_14

[77] T. Bowden, *The /proc filesystem*, 1st ed., 9 June 2009, accessed 4 Sep 2024. [Online]. Available: https://www.kernel.org/doc/Documentation/filesystems/proc.txt

[78] J. Petke, "Constraints: The future of combinatorial interaction testing," in *2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing*, Florence, May 2015, pp. 17–18. [Online]. Available: http://dx.doi.org/doi:10.1109/SBST.2015.11

[79] A. Blot and J. Petke, "Comparing genetic programming approaches for non-functional genetic improvement case study: Improvement of MiniSAT's running time," in *EuroGP 2020: Proceedings of the 23rd European Conference on Genetic Programming*, ser. LNCS, Ting Hu, N. Lourenco, and E. Medvet, Eds., vol. 12101. Seville, Spain: Springer Verlag, 15-17 Apr. 2020, pp. 68–83. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-44094-7_5

[80] ——, "Using genetic improvement to optimise optimisation algorithm implementations," in *23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF'2022*, K. Hadj-Hamou, Ed. Villeurbanne - Lyon, France: INSA Lyon, 23–25 Feb. 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03595447

[81] K. Even-Mendoza, H. D. Menendez, W. Langdon, A. Dakhama, J. Petke, and B. R. Bruce, "Search+LLM-based testing for ARM simulators," in *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, C. Jaspan and R. Kazman, Eds., Ottawa, Canada, April 27-May 3 2025. [Online]. Available: https://solar.cs.ucl.ac.uk/pdf/ICSE_SEIP_2025___SearchSYS_and_ARM.pdf

[82] R. Lorenz, S. H. Bernhart, C. Höner zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker, "ViennaRNA package 2.0," *Algorithms for Molecular Biology*, vol. 6, no. 1, 2011. [Online]. Available: http://dx.doi.org/10.1186/1748-7188-6-26

[83] W. B. Langdon and R. Lorenz, "Improving SSE parallel code with grow and graft genetic programming," in *GI-2017*, J. Petke, D. R. White, W. B. Langdon, and W. Weimer, Eds. Berlin: ACM, 15-19 Jul. 2017, pp. 1537–1538. [Online]. Available: http://dx.doi.org/10.1145/3067695.3082524

[84] ——, "Evolving AVX512 parallel C code using GP," in *EuroGP 2019: Proceedings of the 22nd European Conference on Genetic Programming*, ser. LNCS, L. Sekanina, Ting Hu, and N. Lourenco, Eds., vol. 11451. Leipzig, Germany: Springer Verlag, 24-26 Apr. 2019, pp. 245–261. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-16670-0_16

[85] W. B. Langdon and J. Petke, "Evolving better software parameters," in *SSBSE 2018 Hot off the Press Track*, ser. LNCS, T. E. Colanzi and P. McMinn, Eds., vol. 11036. Montpellier, France: Springer, 8-9 Sep. 2018, pp. 363–369. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-99241-9_22

[86] ——, "Genetic improvement of data gives binary logarithm from sqrt," in *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, R. Allmendinger *et al.*, Eds. Prague, Czech Republic: ACM, 13-17 Jul. 2019, pp. 413–414. [Online]. Available: http://dx.doi.org/10.1145/3319619.3321954

[87] W. B. Langdon, "Genetic improvement of data gives double precision invsqrt," in *7th edition of GI @ GECCO 2019*, B. Alexander, S. O. Haraldsson, M. Wagner, and J. R. Woodward, Eds. Prague, Czech Republic: ACM, Jul. 13-17 2019, pp. 1709–1714. [Online]. Available: http://dx.doi.org/10.1145/3319619.3326800

[88] W. B. Langdon and O. Krauss, "Evolving sqrt into 1/x via software data maintenance," in *9th edition of GI @ GECCO 2020*, B. Alexander, A. S. Brownlee, S. O. Haraldsson, M. Wagner, and J. R. Woodward, Eds. Internet: ACM, Jul. 8-12 2020, pp. 1928–1936. [Online]. Available: http://dx.doi.org/10.1145/3377929.3398110

[89] ——, "Genetic improvement of data for maths functions," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 2, p. Article No.: 7, Jul. 2021. [Online]. Available: http://dx.doi.org/10.1145/3461016

[90] O. Krauss, "oliver-krauss/Replication_GI_Division_Free_Division: Evolving sqrt into 1/x via software data maintenance," May 2020. [Online]. Available: http://dx.doi.org/10.5281/zenodo.3755346

[91] N. Hansen, "CMA-ES source code: Practical hints," 3 June 2011. [Online]. Available: https://cma-es.github.io/cmaes_sourcecode_page.html#practical

[92] W. B. Langdon and R. Poli, "Evolving problems to learn about particle swarm optimisers and other search algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 5, pp. 561–578, Oct. 2007. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2006.886448