Evo* 2025 Late-Breaking Abstracts. Antonio M. Mora *Ed.*, Trieste. 23-25 April 2025.

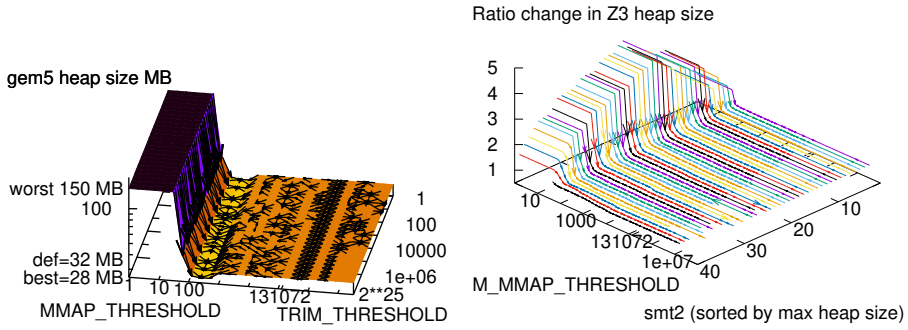
gem5/Z3/gcc/Clang/Redis glibc Heap Fitness Landscapes

William B. Langdon, Justyna Petke, David Clark

W.Langdon@cs.ucl.ac.uk j.petke@ucl.ac.uk david.clark@ucl.ac.uk
CREST, Department of Computer Science,
UCL, Gower Street, London, WC1E 6BT, UK

Abstract. We adapt "The gem5 C++ glibc Heap Fitness Landscape" W.B. Langdon and B.R. Bruce GI@ICSE 2025, to use Valgrind Massif on 1 300 000 line C++ gem5, on 600 000 LOC C++ theorem prover Z3 and benchmarks from SMT-COMP 2024. Showing the memory landscape is far smoother than is commonly assumed and that Magpie and CMA-ES can tune GNU malloc giving 2.4 megabytes reductions in peak RAM use without coding changes. Similar results are given on the GCC and Clang LLVM compilers and 150 000 LOC C Redis key-value database.

Keywords SBSE, computer program tuning, parameter search



Left: Plot of 2 of GNU glibc malloc's 7 parameters. Setting mmap infeasibly small increases the heap size (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. **Right:** Z3 with 40 SMT problems varying only mmap. Again arrows show direction of improvement. Note log scales.

1 Summary

We re-apply genetic improvement [9,6,12,4,11,1,3], to optimise gem5's heap [8] using the newest version of Magpie [2]. Additionally we systematically explore the glibc heap optimisation search landscape of: Microsoft's theorem prover (Z3 [10] 600 000 lines of C++), the GCC and LLVM Clang C++ compilers and the 150 000 LOC C++ Redis Ltd. key-value store database, including also experiments with both Magpie and CMA-ES [5]. Whilst the latter four landscapes are, like gem5, also smooth, they are disappointingly flat and Magpie, by searching for longer, does better than CMA-ES.

2 Introduction

It is sometimes assumed that the fitness landscapes are vast and very difficult. In [8] we showed for a million C++ program, gem5, that this need not be the case and that sometimes real programs give a landscape which is smooth, effectively unimodal and contains many acceptable solutions (left Figure 1). The right hand side of Figure 1 shows this can be true of Z3.

3 glibc Heap 3 Important Parameters

There are 7 GNU malloc tuning parameters. For simplicity we avoid multi-threading, which means only 3 are relevant (see Table 1). Here, like gem5, MMAP_THRESHOLD is dominant.

4 Massif Measuring the Maximum Heap Size

Valgrind's performance tuning tool Massif, can report both dynamic and peak heap usage with high precision. However Massif imposes, particularly if very accurate results are wanted, a runtime overhead. For example, Valgrind 3.23.0 Massif (`--pages-as-heap=yes --peak-inaccuracy=0.0 --time-unit=B`) slowed down gem5 by about 15 fold, Z3 by 5, Clang by 6, and Redis by 4 fold. (On the other hand it had almost no impact on gcc.) In our earlier work [8] we used the GNU malloc_info function, which imposes little overhead. malloc_info requires minor code changes but reports the current heap state, rather than the peak, and so care is needed as to exactly when it is called.

4.1 Massif Fitness Function

The four new benchmarks (Z3, gcc, Clang, Redis) essentially use the same fitness function. The program is run with Massif and the default parameters and then it is run again with Massif and the mutated values of MMAP_MAX, MMAP_THRESHOLD and TRIM_THRESHOLD. Fitness is the ratio of the peak heap usage. If any of the program's outputs are different, the mutant is abandoned.

5 Examples: gem5, Z3, gcc, Clang, Redis

gem5 We described the gem5 benchmark in [8]. The mean distribution of the geometric distribution was corrected to be the same as the glibc defaults. (This was intended, but previously, due to a misunderstanding, it was 2.56× bigger [8].)

Table 1. New Magpie parameter file `mallopt30.txt` E.g. 1st `g[][]` indicates mutations of parameter `MMAP_MAX` are drawn from a geometric distribution, bounded by 0 to 33554432, mean 65536, and default (unmutated) value of 65536.

```
TIMING="run"
M_MMAP_MAX_tune         g[0,33554432,1/65536][65536]
M_TRIM_THRESHOLD_tune   g[0,33554432,1/131072][131072]
M_MMAP_THRESHOLD_tune   g[0,33554432,1/131072][131072]
CLI_PREFIX = ""
CLI_GLUE = "="
```

Z3 The annual international SMT-COMP competition pits SMT solvers against each other on many benchmarks. We choose Certora Prover's QF_UFDTLIA [7], as last year (SMT 2024) Microsoft's Z3 [10][1] did pretty well on it[2]. QF_UFDTLIA contains 76 .smt2 files. We chose the 16^{th} as the best compromise between challenging Z3 and runtime. Like gem5, gcc and Redis, Z3's peak heap memory usage is at the end Figure 2. (Clang's peak heap occurs about 80%.)
gcc We had previously downloaded MySQL[3]. We selected its largest human written C++ source file, ha_innodb.cc, 15 000 LOC. The fitness function compiles it with gcc version 13.3.1 using the same command line and include files as were used to build MySQL (with include files 210 000 lines). To simplify comparing binary object files with and without Massif, we added `-frandom-seed`.
Clang We compiled the same files as gcc with Clang 18.1.8
Redis Ltd. [4]. `redis-server` holds the key-value store. Massif is used to report its heap, whilst `redis-benchmark` exercises it. Thus the fitness function simultaneously runs `redis-server` and `redis-benchmark`. To avoid variability, it was run with a fixed seed, a large (10 000 bytes) SET/GET value and a single client connection. Using Massif limited the number of requests (1024-32 = 992). `redis-benchmark -P 16 -p $PORT -c 1 -d 10000 -n 992`.

6 Search Tools: Grid search, Magpie and CMA-ES

gem5 Two Dimension Grid Search Work on gem5 [8] showed GCC malloc parameter MMAP_MAX, as long as a reasonable value is used, has little impact. Hence in our grid searches it was left at its default value (Table 1). The other two parameters have default values of 2^{17}. In [8] the two dimensional grid search varied both covering all integer and half integer powers of two from 1 to 2^{25} Figure 1 (left). Arrows towards fitness improvements emphasis that although the gem5 fitness landscape is smooth, it is not flat.
One Dimension Grid Search: Z3, gcc, Clang, Redis Figure 1 (left) emphasis that TRIM_THRESHOLD (like MMAP_MAX) has relatively little impact and so for the one dimensional grid searches it was left at its default value and only MMAP_THRESHOLD was varied. However the range was increased to 2^{35}.
Magpie We use Magpie downloaded on 19 Feb 2025[5].
CMA-ES We also use Hansen's CMA-ES[6]algorithm [5].
Both Magpie and CMA-ES were run ten times on Z3, gcc, Clang and Redis.

7 Results

With the corrected means to the distribution of Magpie mutations, see Table 1, the number of Magpie gem5 runs which found good solutions increased from 1/10 to 6/10 (median saving 12.0% ± 0.1%, MMAP_THRESHOLD 300 ± 100).

[1] Z3 downloaded `https://github.com/Z3Prover/z3/` 4 Feb 2025
[2] `https://zenodo.org/records/11061097/files/QF_UFDTLIA.tar.zst` 4 Feb 2025.
[3] `https://github.com/mysql/mysql-server` 12 Mar 2025
[4] `https://github.com/redis/redis/archive/refs/heads/unstable.zip` on 1 Apr
[5] `https://github.com/bloa/magpie`
[6] `https://cma-es.github.io/`

All ten Magpie and all ten CMA-ES runs found solutions which reduced Z3's peak heap usage by 1.5% (Magpie median MMAP_THRESHOLD 30 000 ± 600 and CMA-ES median 50 000 ± 10 000.) See also purple line in Figure 3. Figure 3 +/× shows heap improvement/worsening varies smoothly with MMAP_THRESHOLD for Z3, gcc, Clang and Redis. The mean percentage best improvement found by ten Magpie and ten CMA-ES (both with up to 1000 fitness trials) are shown in Table 2.

Table 2. Mean percentage improvement of ten runs of Magpie and CMA-ES

	Z3	(best)	gcc	(best)	Clang	(best)	Redis	(best)
Magpie	1.5 ± 0.0	1.5	0.0 ± 0.0	0.0	0.0 ± 0.0	0.1	0.3 ± 0.0	0.3
CMA-ES	1.5 ± 0.0	1.5	0.0 ± 0.0	0.0	0.0 ± 0.0	0.0	0.2 ± 0.1	0.3

References

1. Blot, A., Petke, J.: Empirical comparison of search heuristics for genetic improvement of software. IEEE Transactions on Evolutionary Computation **25**(5), 1001–1011 (Oct 2021), http://dx.doi.org/10.1109/TEVC.2021.3070271
2. Blot, A., Petke, J.: MAGPIE: Machine automated general performance improvement via evolution of software. arXiv (4 Aug 2022), http://dx.doi.org/10.48550/arxiv.2208.02811
3. Blot, A., Petke, J.: A comprehensive survey of benchmarks for improvement of software's non-functional properties. ACM Computing Surveys (2025), https://discovery.ucl.ac.uk/id/eprint/10203326/1/main.pdf, in press
4. Bruce, B.R.: The Blind Software Engineer: Improving the Non-Functional Properties of Software by Means of Genetic Improvement. Ph.D. thesis, Computer Science, University College, London, UK (12 Jul 2018), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/bruce_bobby_r_thesis.pdf
5. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2), 159–195 (Summer 2001), http://dx.doi.org/10.1162/106365601750190398
6. Harman, M., et al.: Genetic improvement for adaptive software engineering. In: Engels, G. (ed.) 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'14). pp. 1–4. ACM, Hyderabad, India (2-3 Jun 2014), http://dx.doi.org/10.1145/2593929.2600116, keynote
7. Hozzova, P., Bendik, J., Nutz, A., Rodeh, Y.: Overapproximation of non-linear integer arithmetic for smart contract verification. In: Piskac, R., Voronkov, A. (eds.) Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2023. EPiC Series in Computing, vol. 94, pp. 257–269. EasyChair (2023), https://repositum.tuwien.at/bitstream/20.500.12708/193203/1/Hozzova%20Petra%20-%202023%20-%20Overapproximation%20of%20Non-Linear%20Integer%20Arithmetic%20for...pdf
8. Langdon, W.B., Bruce, B.R.: The gem5 C++ glibc heap fitness landscape. In: Blot, A., Nowack, V., Faulkner Rainford, P., Krauss, O. (eds.) 14th International Workshop on Genetic Improvement @ICSE 2025. Ottawa (27 Apr

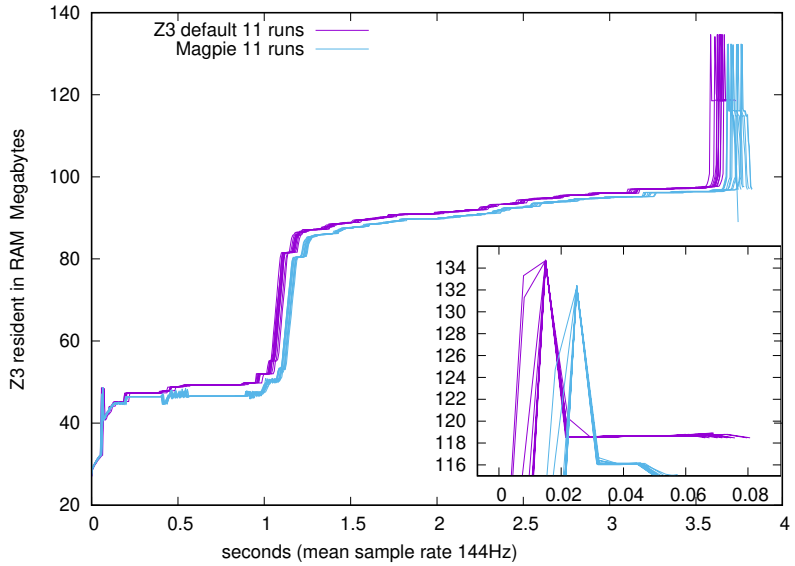


Fig. 2. Z3 use of heap. Magpie chose MMAP_THRESHOLD=29 918. The lower trace shows it saves 2.4MB early in the run but is slightly slower. In the insert the time axis is shifted so the plots synced at the peak value.

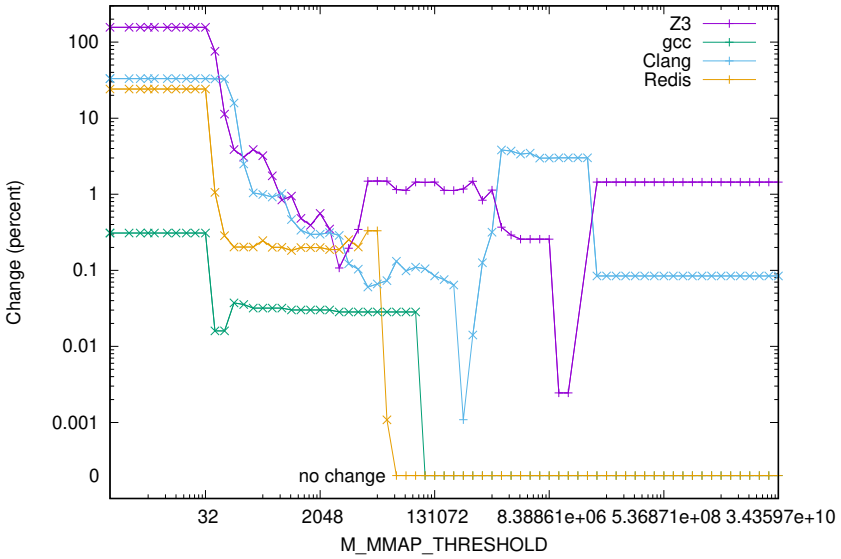


Fig. 3. × percentage increase in peak heap. + percentage reduction. Notice log scales exagerate small differences.

2025), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon_2025_GI.pdf, forthcoming
9. Langdon, W.B., Harman, M.: Optimising existing software with genetic programming. IEEE Transactions on Evolutionary Computation **19**(1), 118–135 (Feb 2015), http://dx.doi.org/10.1109/TEVC.2013.2281544
10. Mendonca de Moura, L., Bjorner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS. LNCS, vol. 4963, pp. 337–340 (2008), http://dx.doi.org/10.1007/978-3-540-78800-3_24
11. Mesecan, I., et al.: HyperGI: Automated detection and repair of information flow leakage. In: Khalajzadeh, H., Schneider, J.G. (eds.) The 36th IEEE/ACM International Conference on Automated Software Engineering, New Ideas and Emerging Results track, ASE NIER 2021. pp. 1358–1362. Melbourne (15-19 Nov 2021), http://dx.doi.org/10.1109/ASE51524.2021.9678758
12. Petke, J., et al.: Genetic improvement of software: a comprehensive survey. IEEE Transactions on Evolutionary Computation **22**(3), 415–432 (Jun 2018), http://dx.doi.org/doi:10.1109/TEVC.2017.2693219