Tag-accessed Memory for Genetic Programming

Alexander Lalejini
Michigan State University
East Lansing, Michigan
lalejini@msu.edu

Charles Ofria
Michigan State University
East Lansing, Michigan
ofria@msu.edu

CCS CONCEPTS

• **Software and its engineering → Genetic programming**;

KEYWORDS

genetic programming, linear genetic programming, tag-based referencing, tags, memory access

ACM Reference Format:
Alexander Lalejini and Charles Ofria. 2019. Tag-accessed Memory for Genetic Programming. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3321892

1 INTRODUCTION

Here, we demonstrate the value of using tags (evolvable labels that can be specified with imperfect matching) to identify memory positions in genetic programming (GP). Specifically, we conducted a series of experiments using simple linear-GP representations on five problems from the general program-synthesis benchmark suite [2]. We show that tag-indexed memory improves problem solving success relative to more traditional, direct-indexed memory.

In traditional software engineering, human programmers create variables with unique names to specify data that they are working with. These variables are inherently associated with locations in memory that are accessed by using the variable's name. This technique for referencing values in memory is intentionally rigid, requiring programmers to precisely name the data they want to reference, and imprecision results in syntactic errors. Many traditional GP systems that give genetic programs access to memory (*e.g.*, indexable memory registers) use similarly rigid naming schemes where memory is numerically indexed, and mutation operators must guarantee the validity of memory-referencing instructions. Interestingly, while exact naming is the most intuitive referencing mechanism for human programmers, evolution in other contexts (such as identifying modules to run [6]) has been shown to be more successful when program references are allowed to be inexact. Here, we present evidence that GP might also be improved by relying on less exacting naming schemes for memory access.

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3321892

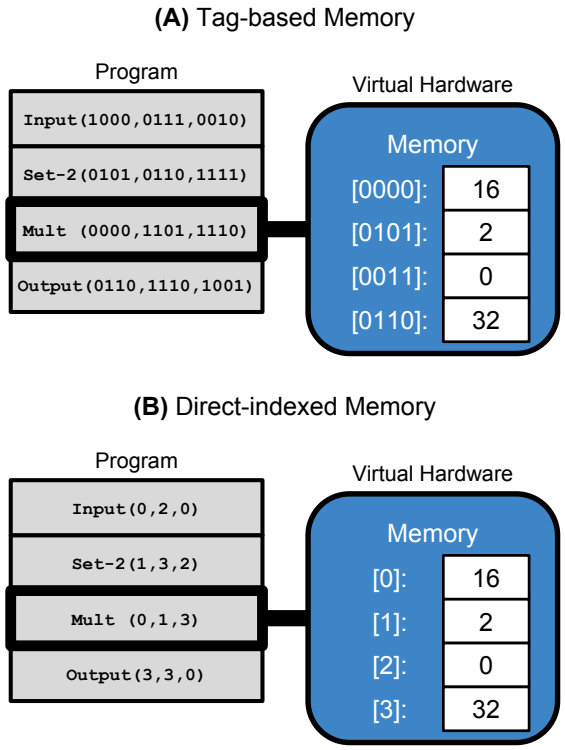

Figure 1: This figure contrasts (A) tag-accessed memory with (B) direct-indexed memory. The programs in (A) and (B) behave identically: both request input to the first memory register, set the second memory register to the terminal value '2', place the result of multiplying the contents of the first two memory registers into the fourth memory register, and output the contents of the fourth register. Here, we show the state of memory after the `Mult` instruction has been executed. Note that all instructions have three arguments; however, not all instructions use all three arguments.

Tags are evolvable labels that give genetic programs a flexible mechanism for specification, originally used by Holland in genetic algorithms ([4]) and refined by Spector *et al.* for GP [7]. To facilitate *inexact* referencing, the similarity (or dissimilarity) between any two tags must be quantifiable; a referring tag can always reference the closest matching referent tag. Here, we continue to expand the integration of tags into linear GP by allowing instructions to use tags to identify positions in memory (as needed for their function). All instructions have three tag-based arguments, each of which is represented as a length-16 bit string and compared using Hamming distance to measure similarity. Our instruction set allows programs to perform basic computations, manipulate memory contents, and

control execution flow (see supplemental material [5] for full details). Programs are executed in the context of a virtual CPU that gives them access to 16 statically tagged memory registers used for storing data for performing computations. Figure 1 contrasts tag-based memory with direct-accessed memory. Tag-based instruction arguments reference the memory position with the *closest matching* tag; as such, argument tags need not *exactly* match any of the tags associated with memory positions.

2 EXPERIMENTAL RESULTS

We compared the performance of our simple linear GP to a variant that replaced the tag-accessed memory with memory indexed with direct arguments (which is more akin to memory access in traditional linear GP [1]). We evolved programs using the lexicase parent selection algorithm [3] to solve five problems from Helmuth and Spector's program synthesis benchmark suite [2]: number IO, smallest, median, grade, and for loop index. For each problem, we added custom instructions to the instruction set that facilitated loading test case inputs into memory and returning program responses. We used the same training and testing sets when evaluating programs as Helmuth and Spector in [2]. We measured performance by counting the number of successful runs (*i.e.*, runs that produced a perfect solution), and we tested for statistical significance using Fisher's exact test (with a significance threshold of 0.05).

For each experimental condition, we evolved 200 replicate populations of 512 individuals (for 100 generations for the number IO problem and 300 generations for all other problems), giving each replicate a unique random number seed. We propagated programs asexually and applied mutations to offspring. We applied single-instruction insertions, deletions, and substitutions at a per-instruction rate of 0.005 each and multi-instruction sequence duplications and deletions at a per-program rate of 0.05. We mutated tag-based arguments at a per-bit rate of 0.005 and numeric arguments at a per-argument rate of 0.005. In preliminary experiments (reported in our supplemental material [5]), we tried a range of argument and tag mutations rates (0.01, 0.001, and 0.005), and our results were consistent. See our online supplemental material ([5]) for source code, details on problem-specific configurations (*e.g.*, program evaluation time, *etc.*), and for our more detailed analyses.

Figure 2 shows the performance of tag-accessed memory and direct-accessed memory across all five problems. In all cases, tag-based instruction arguments (tag-accessed memory) outperformed numeric instruction arguments (direct-accessed memory); differences were significant (Fisher's exact) for every problem (number IO: $p < 2.2e^{-16}$, smallest: $p = 0.0007505$, median: $p = 0.01204$, grade: $p = 0.004591$, and for loop index: $p = 0.0005227$).

3 CONCLUSION

Our preliminary results show the promise of using tag-accessed memory in GP. Further, this technique for labeling and accessing memory is easily applicable to existing GP systems, taking the place of traditional forms of memory access. Here, we limited our experiments to a single linear GP representation and program synthesis problems, however, future work will apply tag-accessed memory across GP systems and across different problem domains. Further, while tag-accessed memory is a promising technique, we still need to explore *why* tag-based referencing works so well for memory access (*e.g.*, is it that programs with tag-based arguments are more robust to mutation than traditional programs?).

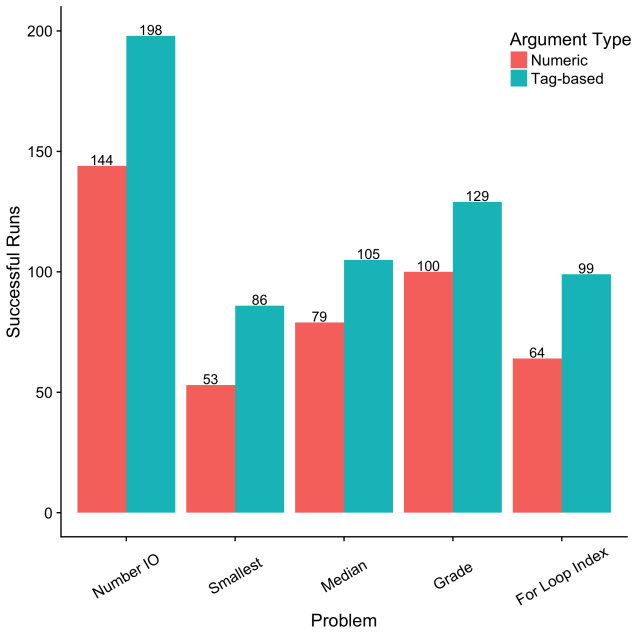

Figure 2: This graph shows the number of successful runs when using our tag-accessed memory versus using traditional direct-indexed memory across five problems (after 100 generations for number IO and 300 generations for all other problems). All differences in performance between tag-accessed and direct-accessed memory are statistically significant.

ACKNOWLEDGMENTS

We extend our thanks to Lee Spector for thoughtful discussion about tag-based referencing (in particular, his idea for a tag space machine). This research has been supported by the National Science Foundation under Grants DGE-1424871 and DEB-1655715.

REFERENCES

[1] Markus Brameier and Wolfgang Banzhaf. 2007. *Linear Genetic Programming.* Springer US, Boston, MA. 315 pages. https://doi.org/10.1007/978-0-387-31030-5
[2] Thomas Helmuth and Lee Spector. 2015. General Program Synthesis Benchmark Suite. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15.* ACM Press, New York, New York, USA, 1039–1046. https://doi.org/10.1145/2739480.2754769
[3] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (10 2015), 630–643. https://doi.org/10.1109/TEVC.2014.2362729
[4] J. Holland. 1993. The effect of labels (tags) on social interactions. *Santa Fe Inst., Santa Fe, NM, Working Paper* (1993), 93âĂŞ10. http://samoa.santafe.edu/media/workingpapers/93-10-064.pdf
[5] Alexander Lalejini. 2019. amlalejini/GECCO-2019-tag-accessed-memory. (Jan. 2019). https://doi.org/10.5281/zenodo.2550905 https://github.com/amlalejini/GECCO-2019-tag-accessed-memory.
[6] Alexander Lalejini and Charles Ofria. 2018. What else is in an evolved name? Exploring evolvable specificity with SignalGP. *PeerJ Preprints 6:e27122v1* (2018), 1–21. https://doi.org/10.7287/peerj.preprints.27122v1
[7] L Spector, B Martin, K Harrington, and T Helmuth. 2011. Tag-based modules in genetic programming. *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation* (2011), 1419–1426. https://doi.org/doi:10.1145/2001576.2001767