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 use of 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 does not substantively affect 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, although exact naming is the most intuitive referencing mechanism for human programmers, evolution in other contexts (such as identifying modules to run [7]) has been shown to be more successful when program references are allowed to be inexact. Beyond computer code, robustness to perturbations is also thought to be important in the evolution of complex biological systems [5].

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

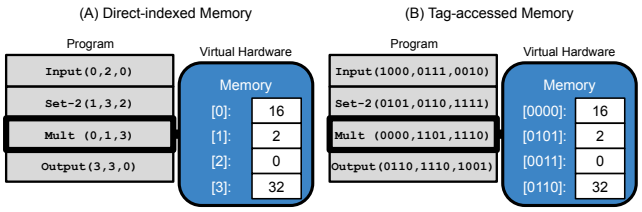

Figure 1: Examples of (A) direct-indexed memory and (B) tag-accessed memory. The programs in (A) and (B) behave identically: both request input to the first memory register, set the second memory register to the terminal value '2', place the result of multiplying the contents of the first two memory 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 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 [8]. 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 [6] for 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-indexed 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. This inexactness makes program phenotypes more robust to minor genetic perturbations, smoothing the genotype-phenotype mapping relative to more traditional memory-indexing techniques.

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

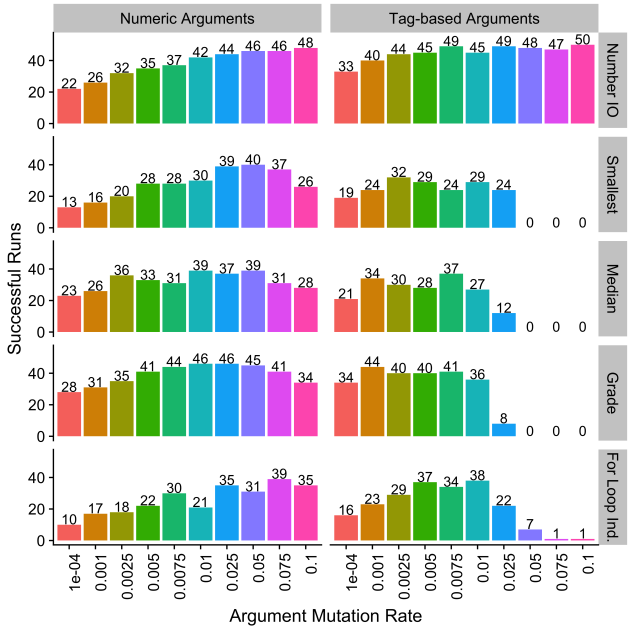

Figure 2: This graph shows the number of successful runs when using our tag-accessed memory (right column) versus using traditional direct-indexed memory (left column) across five problems and ten instruction argument mutation rates (after 100 generations for number IO and 500 generations for all other problems).

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).

For each experimental condition, we evolved 50 replicate populations of 500 individuals (for 100 generations for the number IO problem and 500 generations for all other problems), giving each replicate a unique random number seed. We propagated programs asexually and applied mutations to offspring (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). The relative success of these two memory-indexing techniques is influenced by how (and at what rate) we mutate instruction arguments; as such, we mutated tag-based arguments (per-bit) and traditional arguments (per-argument) at the following ten rates: 0.0001, 0.001, 0.0025, 0.005, 0.0075, 0.01, 0.025, 0.05, 0.075, and 0.1. See our online supplemental material ([6]) 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-indexed memory across all five problems and mutation rates. For each problem, we selected the best (most successful) mutation rate for tag-based arguments and the best mutation rate for traditional arguments. We compared the performance of tag-based arguments and numeric arguments at these 'optimal' mutation rates, and we tested for we tested for statistical significance using Fisher's exact test (with a significance threshold of 0.05). Across all problems, there was no statistically significant difference between tag-based instruction arguments (tag-accessed memory) and numeric instruction arguments (direct-indexed memory).

3 CONCLUSION

Our preliminary experiments show that, under favorable mutation rates, both tag-accessed and direct-indexed memory achieve statistically equivalent performance. Because tag-based instruction arguments index into the *closest matching* memory register, single bit-flip mutations may be neutral (not affecting the program's behavior), which affords programs robustness to minor genetic perturbations. The down-side to a more robust genetic encoding for instruction arguments is that mutations are less able to generate novel phenotypic variation (program behavior). For the relatively simple program synthesis problems used in our experiments, the capacity of our GP system to generate novel phenotypic variation is likely more important than robustness to mutation. Future work will continue to explore the efficacy of tag-accessed memory, supplementing bit-flip mutation operators with more impactful mutation operators that allow tag-mutations to more easily generate novel phenotypic variation. Future work will also investigate the possibility of coevolving register labels (tags) with programs, allowing evolution to adjust the adjacency of memory registers in tag-space.

ACKNOWLEDGMENTS

We extend our thanks to Lee Spector for thoughtful discussion about tag-based referencing (in particular, his idea for a tag space machine). We also thank the Digital Evolution Lab at Michigan State University (MSU) for feedback on this work. This research has been supported by the National Science Foundation through the BEACON center (Cooperative Agreement DBI-0939454) and under Grants DGE-1424871 and DEB-1655715. MSU provided computational resources through the Institute for Cyber-Enabled Research.

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] John. Holland. 1993. The effect of labels (tags) on social interactions. (1993). http://samoa.santafe.edu/media/workingpapers/93-10-064.pdf
[5] Hiroaki Kitano. 2004. Biological robustness. *Nature Reviews Genetics* 5, 11 (Nov. 2004), 826–837. https://doi.org/10.1038/nrg1471
[6] Alexander Lalejini. 2019. Tag-accessed Memory For Genetic Programming. (April 2019). https://doi.org/10.5281/zenodo.2641176 https://github.com/amlalejini/GECCO-2019-tag-accessed-memory.
[7] Alexander Lalejini and Charles Ofria. 2019. What Else Is in an Evolved Name? Exploring Evolvable Specificity with SignalGP. In *Genetic Programming Theory and Practice XVI*, Wolfgang Banzhaf, Lee Spector, and Leigh Sheneman (Eds.). Springer International Publishing, Cham, 103–121. https://doi.org/10.1007/978-3-030-04735-1_6
[8] Lee Spector, Brian Martin, Kyle Harrington, and Thomas Helmuth. 2011. Tag-based modules in genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*. ACM Press, New York, New York, USA, 1419. https://doi.org/10.1145/2001576.2001767