Tag-based module regulation for genetic programming

Alexander Lalejini
University of Michigan
Ann Arbor, Michigan, USA
lalejini@umich.edu

Matthew Andres Moreno
Michigan State University
East Lansing, Michigan, USA
mmore500@msu.edu

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

ABSTRACT

This Hot-off-the-Press paper summarizes our recently published work, "Tag-based regulation of modules in genetic programming improves context-dependent problem solving," published in *Genetic Programming and Evolvable Machines* [1]. We introduce and experimentally demonstrate tag-based genetic regulation, a genetic programming (GP) technique that allows programs to dynamically adjust which code modules to express. Tags are evolvable labels that provide a flexible naming scheme for referencing code modules. Tag-based regulation extends tag-based naming schemes to allow programs to "promote" and "repress" code modules to alter module execution patterns. We find that tag-based regulation improves problem-solving success on problems where programs must adjust how they respond to current inputs based on prior inputs; indeed, some of these problems could not be solved until regulation was added. We also identify scenarios where the correct response to an input does not change over time, rendering tag-based regulation an unnecessary functionality that can sometimes impede evolution. Broadly, tag-based regulation adds to our repertoire of techniques for evolving more dynamic computer programs and can easily be incorporated into existing tag-enabled GP systems.

CCS CONCEPTS

• **Software and its engineering** → *Search-based software engineering*; • **Computing methodologies** → *Artificial intelligence.*

KEYWORDS

tag-based referencing, gene regulation, genetic programming, automatic program synthesis, SignalGP

ACM Reference Format:
Alexander Lalejini, Matthew Andres Moreno, and Charles Ofria. 2022. Tag-based module regulation for genetic programming. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3520304.3534060

1 INTRODUCTION

Genetic programming (GP) applies the principles of evolution to automatically synthesize computer programs instead of writing them by hand. Just as human programmers choose from many different programming languages, each specialized for solving different kinds of problems, GP features many ways of representing evolvable programs. Each representation varies in syntax, organization, interpretation, and evolution. These differences influence the types of programs that can be evolved, shaping a representation's problem-solving range.

Nearly all computer programs must conditionally respond to inputs, and often, the appropriate response to a given input depends on context (*e.g.*, the prior sequence of inputs). Such functionality requires that programs adjust associations between inputs and responses over time. For example, pressing the "equals" button on a calculator triggers different computations depending the preceding sequence of button presses. In [1], we introduce tag-based module regulation for GP, which adds programmatic elements to GP representations that allow us to more easily evolve programs capable of dynamically regulating responses to inputs over time.

2 TAG-BASED MODULE REGULATION

Tag-based module regulation extends existing tag-based naming schemes. Tags are evolvable labels that can be mutated, and the similarity between any two tags can be quantified. Tags are most commonly represented as numeric values [4] or as bit strings [2]. Like traditional naming schemes, tags can specify an arbitrarily large address space. Unlike traditional naming schemes, tags allow for *inexact* addressing. A referring tag refers to a tagged entity (*e.g.*, modules [4] or memory registers [3]) with the *closest matching* tag, ensuring all possible tags are valid references. Moreover, mutations to tags need not change existing referential relationships. As such, mutating tag-based names is not necessarily catastrophic to program functionality (as it would be in traditional naming schemes).

Tag-based module regulation allows programs to "promote" or "repress" code modules (Fig. 1). We describe our implementation of tag-based regulation in the context of a module-based linear GP system, SignalGP [2]), but our overall approach is applicable to tag-enabled GP representations. Briefly, SignalGP programs comprise tag-addressed modules (*i.e.*, functions), each of which contains a sequence of instructions. Each instruction has arguments, including an evolvable tag-based argument that can be used to call a tag-addressed module. When an instruction attempts to call a module, all modules in the program are ranked according to the tag-match score between the instruction and module tags (based on tag similarity), and the module with the best tag-match score is chosen.

To allow for tag-based module regulation, we added a "regulatory modifier" to all tag-addressed modules. This value adjusts how well the module's tag matches to referring tags, modifying the likelihood it will be referenced. We additionally added a set of promoter and repressor instructions that, when executed, adjust a target module's regulatory modifier: promoter instructions make a target module

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3534060

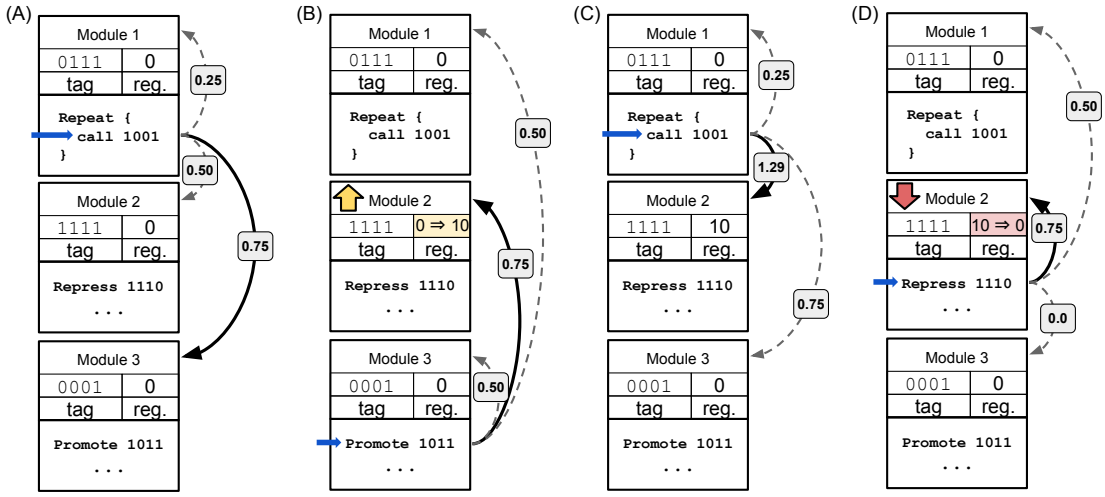

Figure 1: Tag-based genetic regulation example (adapted from [1]). This example depicts a simple oscillating regulatory network instantiated using tag-based regulation. In this example, tags are length-4 bit strings. The "raw" match score between two tags equals the number of matching bits between them. Regulation (reg.) modifies match scores for "`call`" instructions according to Equ. 1 in [1]. First (A), the `call 1001` in Module 1 executes, triggering Module 3. Next (B), Module 3 is executed, promoting Module 2. After Module 3 returns, the `call 1001` in Module 1 executes again (C); however, Module 2's promotion causes it to be triggered instead of Module 3. Finally (D), Module 2 executes and represses itself, resetting its regulatory modifier.

more likely to be referenced, and repressor instructions have the opposite effect. See Section 3 of [1] for further details.

3 SUMMARY OF RESULTS

First, we assessed our implementation of tag-based module regulation: Can we evolve programs capable of dynamically adjusting their response to environmental conditions over time? And, can the addition of tag-based module regulation improve problem-solving success on context-dependent problems? We addressed these questions using the signal-counting and contextual-signal problems, two diagnostic tasks that require context-dependent responses to input signals. The signal-counting problem requires programs to continuously change their response to an environmental signal, producing a different designated output each time the signal is repeated. The contextual-signal problem requires programs to respond to a pair of input signals such that the first input signal determines the correct output in response to the second signal.

For both the signal-counting and contextual-signal problems, we observed the evolution of adaptive tag-based module regulation, and we found that the addition of tag-based module regulation improved problem-solving success (Table 4 and Fig. 5 in [1]). Moreover, for some problem difficulty levels, solutions only evolved when tag-based module regulation was enabled.

Next, we evaluated tag-based module regulation on the Boolean-logic calculator problem, which requires that programs implement a push-button calculator capable of performing 10 bitwise logic operations (Table 3 in [1]). Tag-based module regulation improved problem-solving success when performing the correct logic operation required programs to recall a previous input signal (Fig. 6 in [1]). However, when we reduced the context dependence of outputs (*i.e.*, inputs given in postfix notation), tag-based module regulation no longer improved problem-solving success (Fig. 9 in [1]).

Finally, we used the independent-signal diagnostic task to demonstrate a scenario where tag-based regulation is not useful. The independent-signal problem requires programs to execute a unique response for each of sixteen different input signals; because input signals are unique, programs do not need to adjust their responses to any particular signal over time. We found that erroneous regulation hindered task generalization on this problem (Fig. 8 in [1]); that is, programs might produce perfect output for one sequence of inputs, but for another input sequence, erroneous module regulation caused programs to produce incorrect output.

Overall, our results demonstrate that tag-based module regulation can improve GP systems' capacity to evolve more dynamic computer programs. Future work will apply tag-based module regulation in more contexts and investigate ways to scale up tag-based regulation because we found that evolved regulatory networks became increasingly brittle as they grew to large sizes.

REFERENCES

[1] Alexander Lalejini, Matthew Andres Moreno, and Charles Ofria. 2021. Tag-based regulation of modules in genetic programming improves context-dependent problem solving. *Genetic Programming and Evolvable Machines* (July 2021). https://doi.org/10.1007/s10710-021-09406-8

[2] Alexander Lalejini and Charles Ofria. 2018. Evolving event-driven programs with SignalGP. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18*. ACM Press, Kyoto, Japan, 1135–1142. https://doi.org/10.1145/3205455.3205523

[3] Alexander Lalejini and Charles Ofria. 2019. Tag-accessed memory for genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '19*. ACM Press, Prague, Czech Republic, 346–347. https://doi.org/10.1145/3319619.3321892

[4] 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, Dublin, Ireland, 1419. https://doi.org/10.1145/2001576.2001767