# Numerical-Node Building Block Analysis of Genetic Programming with Simplification

Phillip Wong, Mengjie Zhang
Victoria University of Wellington, P.O. Box 600,
Wellington, New Zealand
{phillip, mengjie}@mcs.vuw.ac.nz

## ABSTRACT

This paper investigates the effects on building blocks of using online simplification in a GP system. Numerical nodes are tracked through individual runs to observe their behaviour. Results show that simplification disrupts building blocks early on, but also creates new building blocks.

## Categories and Subject Descriptors

I.2.m [**Artificial Intelligence**]: Miscellaneous

## General Terms

Performance, Experimentation

## Keywords

Genetic Programming, Simplification, Code Bloat, Numerical-nodes

One current problem in GP is that of *code bloat* [1]. Genetic programs tend to grow very quickly in size (mostly through redundant code), easily out-pacing the improvements in program fitness as evolution progresses. Simplification [2] is one approach to combating code bloat and has been tested on several GP tasks, showing that simplification can significantly improve the efficiency of the GP system (i.e. training time and program size), as well as the effectiveness (fitness of solutions) [2]. However, simplification may also remove useful building blocks within the GP population. This work aims to determine whether simplification disrupts or creates building blocks during GP evolution.

Simplification is the process of directly removing redundancy from a program, leaving behind a smaller, semantically equivalent program (yields the same outputs given the same inputs). For the purposes of this paper, we will use an *Algebraic Simplification* method described in GECCO '06 [2]. This method uses a set of algebraically based rules, as well as hashing techniques.

We track a simple form of building block (numerical-nodes) throughout a GP run in order to observe their behaviour under simplification. These are simply the numerical terminal nodes in GP programs. In standard GP, these are usually regarded as "constant". However, in GP systems using simplification, this is no longer true as they can be altered by simplification. Essentially, numerical-nodes can exhibit nearly all the characteristics of larger building blocks, but with reduced complexity, allowing for easier tracking and analysis. They provide a good "first step" into analysing the influence simplification has on GP building blocks. We will analyse more "general" building blocks in future work.

For experiments, we used the symbolic regression task $11x + 50.0$ for 200 points over the range $[-10.0, 10.0]$. This task put emphasis on the use of numerical-nodes. Mean square error over all the points is used as the fitness measure. In both "standard GP" and "GP with simplification", tree-based programs are used. The ramped half-and-half method was used for program generation. Proportional selection and reproduction crossover and mutation were used as the genetic operators. The terminal set consists of a single variable and a number of randomly generated numerical-nodes in the range of $[-1.0, 1.0]$. The use of a small range is used in order to increase the difficulty of the problem. The function set contains the four arithmetic functions: $+$, $-$, $\times$, $\div$ (protected division). The following parameters were used for both GP with or without simplification: Mutation 30%, Elitism 10%, Crossover 60%, Population Size 500, Generations 100, Max. Depth Limit 6. Additionally for GP with simplification, several frequencies of how often simplification was applied were used: none, every 1, every 5, every 10. *Hash order (p)* for the hashing process is set to 1000077157, and the *constant precision ($\delta$)* is set to 1000000.

It was found that, in general, simplification does in fact disrupt existing building blocks, which can make it more difficult for a system to find a highly fit solution. We also found that simplification was able to create new numerical-nodes, some of which were propagated by the GP system to create better solutions. In most cases, new building blocks were able to be constructed, and GP systems using simplification were able to recover from the disruption of building blocks and even outperform the standard GP. Results suggest that system performance for GP with simplification relies on these building blocks being created.

It is clear that these interactions need to be further investigated. We will analyse more complex forms of building blocks in future works.

[1] Terence Soule and James A. Foster and John Dickinson, Code Growth in Genetic Programming, in *Genetic Programming 1996*, pages 215–223, MIT Press.

[2] Phillip Wong and Mengjie Zhang, Algebraic simplification of GP programs during evolution, in *GECCO 2006*, pages 927–934, ACM Press.


*GECCO'07,* July 7–11, 2007, London, England, United Kingdom.
ACM 978-1-59593-697-4/07/0007.