Parallel Genetic Programming

Abiyaz Chowhury Allen Kim

***Abstract*—Genetic programming is a technique in the field of genetic algorithms that evolve computer programs themselves. Programs are encoded in some computational representation that are then run through the standard genetic operations (selection, crossover, mutation). However, the computational cost to train genetic programs grows rapidly as the complexity of the problem increases. We aim to implement a highly parallel version of genetic programming that will scale efficiently.**

***Index Terms*—parallel, genetic, programming**

I. Introduction

Genetic algorithms have primarily seen use in optimizing numerical parameters. However, genetic programming has not seen as much mainstream use in the past due to its severe computational costs. Genetic programming aims to evolve the computer program themselves over time rather than various function parameters [1]. By having programs themselves be the population, genetic programming ultimately aims to have computer-generated programs that show distinct characteristics from human written programs and ideally, efficient as well.

The main problem with genetic programming is the fact that in practice, it is only able to solve small problems. The sheer number of possible programs that exist makes it difficult to even create simple programs. However, with the growth in modern computing power and the use of parallelism, genetic programming is beginning to seem more practical.

II. Prior Work

One main topic in prior work in genetic programming deals with how to represent the programs themselves structurally. The simplest idea was to use a forest of parse trees since compilers are used to dealing with such structures for programs, but they tend to be much slower to analyze. Other data structures such as directed acyclic graphs [3] were later proposed to allow for greater efficiency, but we focused on parallel distributed genetic programming (PDGP) [2].

In PDGP, programs are encoded as grids in which each node can be non-terminal or terminal. Non-terminal nodes represent different functions that take in other nodes as arguments while terminal nodes represent the input to the program. There are many variations to the grid, but we chose one of the simplest representations. Akin to a feed forward neural network, we considered grids that had connective edge between adjacent layers. Thus, our programs would propagate values upward from bottom to top through a series of nodes. This was still an efficient representation as nodes can be reused by the layers above, but was also simple enough to understand and implement. With the identity node, any grid could be represented in this way given sufficient rows and columns.

Regarding the parallel implementations of genetic programming, most prior work splits up the population among the processors and evaluate fitness in parallel [4]. We aimed to implement these core ideas in our program as well.

III. Overview

For our project, we implemented a parallelized version of PDGP, which was built to solve arbitrary combinatorial circuit problems. We allow the use of AND, OR, NOT, NAND, NOR, and I (identity) as our non-terminals. We also use x_1 to x_n for our input. Given a function we want to learn, we use our program to find circuits that can solve it. We test our program on solving XOR as well as solving the even-3 parity problem.

IV. Description of Implementation

A. Random Number Generator

B. Random Graph Generation

C. Genetic Operators

1) *Crossover:*
2) *Mutation:*
3) *Selection:*

D. Evaluation and Fitness

E. Multi-start Evaluation

V. Results

VI. Future Work

References

[1] McPhee, Nicholas Freitag, Riccardo Poli, and William B. Langdon. Field guide to genetic programming. (2008).

[2] Poli, Riccardo. Parallel distributed genetic programming. University of Birmingham, Cognitive Science Research Centre, 1996.

[3] S.Handley. On the use of a directed acyclic graph to represent a population of computer programs. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, pages 154-159, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[4] Kilian Stoffel and Lee Spector. High-performance, parallel, stack-based genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, Genetic Programming 1996: Proceedings of the First Annual Conference, page 224, Stanford University, CA, USA, 28-31 July 1996. MIT Press.

[5] B. Edmonds, Meta-genetic programming: Co-evolving the operators of variation, CPM Report No.: 98-32. Centre for Policy Modelling, Manchester Metropolitan University. http://www.cpm.mmu.ac.uk/cpmrep32.html, 1998.