# Parallel Computing Project Proposal

Morgan R. Frank

October 18, 2015

Genetic programming (GP) is an evolutionary algorithm for creating models that predict an output variable given input data. Like all evolutionary algorithms, GP utilizes biological genetics as a paradigm for problem solving by treating syntax trees as genomes and their ability to match a target variable given input data as the phenotype. While many different learning algorithms exist for pursuing this same goal, genetic programming, and evolutionary algorithms in general, are appealing because of the widely understood principles they are based on. Namely, almost everyone has taken a biology class that highlighted the role of genetic crossover and genetic mutation as the driving mechanisms for solving one of the greatest problems: how to be a better organism.

The algorithm begins with an initial population of syntax trees and a fitness function which takes a syntax tree as input and produces a nonnegative real number representing the tree's ability to reproduce a desired output. In the most basic version of GP, new generations of syntax trees are created from the previous generation of trees using a combination of genetic mutation and crossover after applying selection pressure through the fitness scores. Mutation is implemented by randomly changing a node on the syntax tree, while crossover is implemented by choosing two "fit" parent trees and swapping randomly selected subtrees. More complicated GP implementations usually apply additional selection pressure for the complexity of the model (ie. the depth of the syntax tree) and the age of a solution (ie. how many generations a syntax tree has been in the population). Combining all of these selection pressures yields an algorithm that produces accurate and simple models, while maintaining the appropriate proportion of exploration and exploitation within the GP population.

One can emphasis these two desirable dynamics by making use of parallel computation in a nontrivial way. If we run independent GP algorithms on each core (or node, if using a cluster) and occasionally allow migration of fit syntax trees from their population, or "island", to another parallel population, then we can potentially improve the aggregate performance of the island model GP by allowing individual islands to optimize an area of search space, while still implementing some exploration through this migration process. This model is highly scalable since we can add an arbitrary number of islands, and adding additional islands only serves to improve the aggregate performance of the island model

GP.

I plan to implement the island model GP in Julia. The resulting implementation should allow for easy transition from a multi-core computer to a distributed compute cluster. GP is an intriguing and flexible learning algorithm for a variety of problems, from symbolic regression to evolving algorithms, and I believe my project could lead to an interesting addition to the Julia library.